Diss. ETH 5941

Informationsverdichtung von Modellen in der Aussagenlogik und das P = NP - Problem

ABHANDLUNG

zur Erlangung
des Titels eines Doktors der Mathematik
der
EIDGENOESSISCHEN TECHNISCHEN HOCHSCHULE
ZUERICH

vorgelegt von
Karl Lieberherr
dipl.Math.ETH
geboren am 27. August 1948
von Nesslau (SG)

Angenommen auf Antrag von
Prof.Dr.E.Engeler, Referent
Prof.Dr.E.Specker, Korreferent

1977

```
procedure SETZE(l,s,WIDERSPRUCH);
begin
   el := {l} ; WIDERSPRUCH := false ;
   repeat
      waehle ein Element ll aus el ;
      el := el - {ll} ;
      seien kpll die Klauseln in s,
      welche den Literal ll enthalten ;
      for jede Klausel c in kpll do entferne c aus s;
      seien knll die Klauseln in s, welche den Literal ll´
      enthalten;
      for jede KLausel c in knll do
      begin
         streiche den Literal ll´ in der KLausel c von s;
         if c enthaelt genau einen Literal l2 then
         (* l2 heisst eindeutig bestimmt *)
         if l2´ in el then WIDERSPRUCH := true else
         el := el + {l2}
      end
   until el={} ;
end;
```

Informell ist eine Superresolvente sr einer KNF s eine Klausel, fuer die folgendes gilt: Seien alle Literale in sr mit Hilfe von SETZE so gesetzt, dass sr unerfuellt ist. Die Variablen, die zu anderen Literalen als denen in sr gehoeren, seien noch nicht interpretiert. Dann darf keine Klausel in s unerfuellt sein. Jetzt muss es eine Variable v geben, deren beide Interpretationen durch Konsequenzenbildung (d.h. durch Beruecksichtigen der eindeutig bestimmten Literale) eine unerfuellte Klausel entstehen lassen. Die Variable v nennen wir lernend.

Nun geben wir die exakte Definition fuer den allgemeineren Begriff Superresolvente k-ten Grades.

Definition 1.1.1 : Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente k-ten Grades (k>1) einer KNF s, wenn man SETZE(l(i),s,w) der Reihe nach fuer i = 1, 2, ... n ausfuehren kann und immer w=false erhaelt, und wenn dann

1. k Variablen v(1), v(2), ... v(k) existieren, so dass fuer alle Folgen vv(1), vv(2), ... vv(k) (vv(j) in {v(j),v(j)´} (1<j<k)) die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k ausgefuehrt, immer w=true liefert und
2. fuer jedes k1<k eine Folge vv(1), vv(2), ... vv(k1) existiert, so dass die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k1 ausgefuehrt, w=false liefert.

Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente einer KNF s, wenn man SETZE(l(i),s,w) der Reihe

Sehr herzlich moechte ich Herrn Prof. Dr. E. Engeler fuer die stets wohlwollende Betreuung meiner Arbeit und Herrn Prof. Dr. E. Specker fuer die Uebernahme des Korreferates danken. Meiner Frau Dr. R. Lieberherr, meinen Kollegen H.R. Thomann, Dr. R. Schoenberger, E. Graf, M. Wietlisbach und M. Fuerer danke ich fuer ihre Kritik und ihre Anteilnahme an meiner Arbeit. Dankbar bin ich A. Mueller fuer das Trennungsprogramm, das er mir zur Verfuegung stellte.

Weiterhin danke ich allen Professoren und den Sekretaerinnen des Instituts fuer Informatik und meinen Institutskollegen fuer das angenehme Arbeitsklima, das meine Arbeit indirekt unterstuetzt hat.

Abstract:

The thesis deals with the problem of deciding, whether a given formula of the propositional calculus is satisfiable. In the first part a deduction rule, called Superresolution, is introduced and compared with known proof-systems and decision procedures. In the second part probabilistic algorithms for NP-complete languages are investigated.

Keywords:
Superresolution, decision procedures for Satisfiability, polynomial reduction of proof-systems, length of proofs, lower bounds, probabilistic algorithms for NP-complete languages.

Zusammenfassung

In dieser Arbeit wird die Komplexitaet des Entscheidens, ob eine konjunktive Normalform (KNF) der Aussagenlogik erfuellbar ist, analysiert. Dieses Entscheidungsproblem hat eine zentrale Stellung in einer grossen Klasse von kombinatorischen Entscheidungs- resp. Optimierungsproblemen.

Die bisher untersuchten Beweissysteme fuer die unerfuellbaren KNF haben lokalen Charakter. In der Arbeit wird ein Beweissystem Superresolution eingefuehrt, das als global bezeichnet werden darf, d.h. ein Schluss kann von der ganzen bisherigen Formel abhaengen. Superresolution wird verglichen mit einem auf dem Gebiete des automatischen Beweisens haeufig verwendeten Beweissystem, naemlich mit Resolution und deren Abarten. Weil Resolution lokal ist, so ist es einleuchtend, dass eine globale Schlussregel wie Superresolution kuerzere Beweise erlaubt als Resolution. Dies wird in der Arbeit auch bewiesen. Dafuer ist der Aufwand fuer das Finden, resp. Ueberpruefen eines Beweisschrittes groesser als bei Resolution. Beide Aufgaben sind jedoch linear loesbar. Superresolution ist eine grosse Einschraenkung von Resolution, denn keine Input-Resolvente kann eine Superresolvente sein. Deshalb hat Superresolution gegenueber Resolution wesentliche Vorzuege, die in der Arbeit untersucht werden.

Um die Komplexitaet von Superresolution zu analysieren, fuehrt man einen Entscheidungsalgorithmus SR fuer KNF ein, der nur Superresolutionsbeweise in einer kurzen Normalform erzeugen kann. Es stellt sich dabei heraus, dass Superresolution und Resolution polynomial aequivalent sind. Ausserdem wird fuer eine spezielle Art von Superresolution eine exponentielle untere Schranke bewiesen. Neben anderen Vorzuegen hat SR die Tendenz, "einfache" Formeln effizient zu behandeln.

Im zweiten Teil der Arbeit werden probabilistische Entscheidungsalgorithmen fuer NP-vollstaendige Sprachen untersucht. Probabilistische Algorithmen sind keine Algorithmen im eigentlichen Sinn, denn sie verwenden bei ihren Berechnungen Zufallsprozesse und koennen deshalb falsche Entscheide liefern. Wenn die Fehlerwahrscheinlichkeit aber mit relativ wenig Rechenaufwand gegenueber den bekannten deterministischen Verfahren beliebig klein gemacht werden kann, sind probabilistische Algorithmen fuer praktische Anwendungen sehr interessant.

In der Arbeit wird der Begriff "Erfuelltheitsgrad" fuer KNF eingefuehrt und auf NP-vollstaendige Sprachen erweitert. Die Komplexitaet von probabilistischen Algorithmen fuer eine Sprache S haengt direkt mit dem Erfuelltheitsgrad von S zusammen. Die Relation wird durch eine Formel beschrieben, die ausdrueckt: Je groesser der Erfuelltheitsgrad ist, desto effi-

zientere Algorithmen resultieren. Deshalb wird in der Arbeit versucht, den Erfuelltheitsgrad moeglichst gross zu machen. Es wird behauptet und durch verschiedene Ueberlegungen und Hilfssaetze begruendet, dass dies mit wenig Aufwand moeglich ist, beim Versuch, es zu beweisen, stoesst man aber auf erhebliche Schwierigkeiten.

Inhaltsverzeichnis

Bezeichnungen

Dieser Text wurde mit Hilfe einer computergesteuerten Schreibmaschine geschrieben, welche die ueblichen mathematischen Symbole nicht im Zeichensatz hat. Deshalb verwenden wir folgende Bezeichnungen:

abs	abs(a) : absoluter Betrag von a
binomial	binomial(n,k) ist der Binomialkoeffizient n tief k
card	card(a) : die Maechtigkeit der Menge a
in	a in B : a ist ein Element von B
trunc	Fuer nicht negative reelle Zahlen a ist trunc(a) die groesste natuerliche Zahl, die kleiner oder gleich a ist.
**	Potenzierung
{}	leere Menge
+	Vereinigung von Mengen (und Addition)
*	Durchschnitt von Mengen (und Multiplikation)
#	Ende eines Beweises
v	Disjunktion (oder)
&	Konjunktion (und)
´	Negation (a´ ist die negierte Variable a)

Tiefstellungen werden durch eckige oder runde Klammern dargestellt, z.B. a[1] oder a(1).

INFORMATIONSVERDICHTUNG VON MODELLEN IN DER AUSSAGENLOGIK UND DAS P = NP - PROBLEM

****

Uebersicht

****

In dieser Arbeit untersuche ich einen Entscheidungsalgorithmus SR und einige seiner Abwandlungen fuer das Erfuellbarkeitsproblem (ERF) der konjunktiven Normalformen (KNF) der Aussagenlogik. Dabei diente mir folgende Frage als Leitmotiv : Sei s eine erfuellbare KNF mit Modell M und r eine Klausel, die eine Folgerung von s ist. Wie kann man r so waehlen, dass r viele Informationen ueber M enthaelt, d.h. dass M relativ viele Literale in r erfuellt ? Die Modellinformation Inf(r,M) von r in bezug auf M ist der Quotient der Anzahl durch M erfuellter Literale und der Literalanzahl von r. Wenn man die Erzeugung von Klauseln r mit Inf(r,M) groesser als 1/2 iteriert, erhaelt man zuverlaessige Informationen ueber M durch blosse zufaellige Auswahl von Literalen in den erzeugten Klauseln. Diese Tatsache laesst sich dazu verwenden, probabilistische Algorithmen [RAB76,SOL76] zu entwickeln, die Modelle mit vorgegebener Fehlerwahrscheinlichkeit finden, und zwar rascher als die trivialen deterministischen Algorithmen.

Uebersicht ueber die Resultate der Arbeit:

1. Ich habe ein vollstaendiges Beweissystem (Superresolution) fuer unerfuellbare KNF entwickelt, das stets besser ist als Resolution. D.h. fuer jeden Resolutionsbeweis, der nicht trivial ist (d.h. die leere Klausel ist nicht die erste erzeugte Klausel), existiert ein Superresolutionsbeweis, der weniger Klauseln enthaelt. (Es ist zu erwarten, dass im verkuerzten Beweis die Anzahl Anwendungen der Schlussregel "Superresolution" meistens etwa die Quadratwurzel der Anzahl Anwendungen der Schlussregel "Resolution" im unverkuerzten Beweis ist.)
   Eine Anwendung der Schlussregel "Superresolution" auf eine Formel s beruecksichtigt meistens die "ganze" Formel s. Deshalb kann Superresolution als globales Beweissystem betrachtet werden. Das Ueberpruefen eines Superresolutionsbeweisschrittes ist in linearer Zeit moeglich. Superresolution ist in folgendem Sinn staerker als Resolution: Man braucht mindestens 3 Anwendungen von Resolution, um ir-

gendeine Klausel (ausser die leere) zu erhalten, die mittels Superresolution mit einer einzigen Anwendung der Schlussregel erzeugbar ist. Nach meinen Literaturkenntnissen (siehe Bibliographie Teile A und D) sind diese Resultate neu, obwohl schon in verschiedenen Arbeiten nach einem solchen Ergebnis gesucht wurde. Superresolution ist auch besser als die bekannten Verfeinerungen von Resolution wie: semantische Resolution, Hyperresolution, set-of-support-Resolution, Lockresolution, lineare Resolution, Resolution mit Mischen etc. Ich habe Superresolution zu einem Beweissystem "erweiterte Superresolution" ergaenzt und bewiesen, dass "erweiterte Superresolution" stets besser ist als "erweiterte Resolution".

2. Resolution und Superresolution sind zwar als Beweissysteme fuer unerfuellbare KNF polynomial aequivalent, doch kann ich in Abschnitt 1.5 beweisen, dass von einem bestimmten algorithmischen Gesichtspunkt her Superresolution exponentiell besser ist als Resolution.

3. Weiter beweise ich in Abschnitt 1.8 fuer bestimmte Mengen von KNF [TS68, COO76], dass SR auf diesen Mengen nur polynomialen Aufwand hat. Hingegen haben gewisse, nicht-triviale klassische Entscheidungsalgorithmen (Davis-Putnam, regulaere Resolution) fuer diese Klasse erwiesenermassen exponentiellen Aufwand.

4. Es zeigt sich in Abschnitt 1.7, dass auf den bekannten Teilmengen von ERF (Horn-KNF, Krom-KNF, stark erfuellbare KNF), die polynomial entscheidbar sind, SR uniform die Entscheidung in polynomialer Zeit liefert. Dabei werden die eingegebenen KNF nicht in verschiedene Typen eingeteilt, sondern alle werden nach derselben universellen Methode behandelt. Dies deutet an, dass SR die Tendenz hat, sich auf "einfachen" Formeln guenstig zu verhalten. Damit wird die Effizienz von SR noch weiter unterstrichen.

5. Weiter zeige ich, wie Algorithmus SR dazu verwendet werden kann, Boolesche Funktionen beschleunigt zu berechnen (Abschnitt 1.9).

6. Ich zeige, dass die Mittelwertuntersuchung von gewissen Resolutionsbeweissystemen ein System ergibt, das eine besonders gute Verdichtung von Modellinformation garantiert (Abschnitt 2.4).

7. Sei s eine erfuellbare KNF mit Modell M, und der Erfuelltheitsgrad eq(s,M) sei der Bruchteil der von M erfuellten Literale in s. Ich gebe NP-vollstaendige Teilmengen Q von ERF an, so dass fuer alle KNF q in Q und fuer alle Modelle M von q der Erfuelltheitsgrad beliebig nahe 1 ist (Abschnitt 2.3).

8. Es gibt polynomiale Verfahren, denen eine Erfolgswahrscheinlichkeit d zugeordnet werden kann, fuer die gilt: Je groesser d ist, desto effizientere Erkennungsalgorithmen mit vorgegebener Fehlerwahrscheinlichkeit koennen fuer eine gewisse NP-vollstaendige Sprache angegeben werden. d = 1 wuerde bedeuten, dass P = NP ist. Ich zeige, dass d $\geq$ 2/3. Das bedeutet, dass nur O(1.5**n) Interpretationen konstruiert werden muessen, damit eine beliebig kleine Fehlerwahrscheinlichkeit erhalten wird (Abschnitte 2.2, 2.3).

9. Ich zeige fuer eine NP-vollstaendige Teilsprache Q von ERF, dass fuer jedes q in Q der Aufwand fuer eine Variante von SR von der Ordnung O(p(Laenge(q))*2**(2*n/3)) ist. Dabei ist n die Anzahl Variabeln in q und p ist ein Polynom kleinen Grades. Kuerzlich haben Tarjan und Trojanowski in [TAR76] einen O(2**(n/3)) Algorithmus fuer eine andere NP-vollstaendige Sprache angegeben.
Diese beiden Resultate zeigen, dass sogar fuer gewisse NP-vollstaendige Sprachen Algorithmen existieren, die im schlimmsten Fall wesentlich besser sind als die naheliegenden Aufzaehlungsalgorithmen (Abschnitt 1.14).

10. Unter Verwendung eines Resultates von Galil [GAL76] ueber Aufzaehlungsalgorithmen zeige ich in Abschnitt 1.13 exponentielle untere Schranken fuer eine spezielle Art von Superresolution.

11. Als Hauptresultat meiner Arbeit entwickle ich in Abschnitt 2.6 einen Entscheidungsalgorithmus fuer ERF. Er hat vermutlich polynomialen Aufwand, wobei aber die Entscheidung nur mit beliebig kleiner Fehlerwahrscheinlichkeit gefaellt wird. Erweist sich meine Vermutung als richtig, so ist fuer praktische Zwecke P=NP, d.h. dass jedes Problem in NP in polynomialer Zeit mit beliebig kleiner Fehlerwahrscheinlichkeit geloest werden kann. Es gibt bereits Resultate in dieser Richtung : Solovay und Strassen [SOL76] haben fuer die Primzahlen (die, wie auch ihr Komplement, in NP sind) sogar einen linearen Algorithmus angegeben, der die Entscheidung mit beliebig kleiner Fehlerwahrscheinlichkeit faellt.

Ø. Grundbegriffe

Die Komplexitaetstheorie beschaeftigt sich unter anderem mit der Frage : Welche Funktionen sind praktisch berechenbar ? Bevor wir diese Frage beantworten, benoetigen wir einige Definitionen.

Als Computermodell verwenden wir die Direktzugriffmaschine (DZM), wie sie in [AHO75,Seite 5] beschrieben ist. Dabei setzen wir voraus, dass die Ausfuehrung einer beliebigen Anweisung eine Zeiteinheit benoetigt.

Sei B* die Menge der endlichen Worte ueber dem Alphabet B={Ø,1}. (Jedes Element von B* ist als Codierung eines Objektes zu betrachten.) Eine Funktion f : B* -> B* heisst partiell berechenbar, wenn es ein Programm p fuer die DZM gibt, das fuer jedes x im Definitionsbereich von f nach endlich vielen Schritten stoppt, nachdem es f(x) auf das Ausgabeband geschrieben hat. Die Laenge l(x) eines Elementes x in B* ist die Anzahl Zeichen, die x enthaelt.

Sei f eine durch das Programm p berechnete Funktion. Wir betrachten die Menge c(n) saemtlicher Elemente x im Definitionsbereich von f, welche die feste Laenge l(x)=n haben. Sei s(x) die Zeit, die p benoetigt, um f(x) zu berechnen. Wir definieren

$$K[f,p](n) = \max[x \text{ in } c(n)]\ s(x)/l(f(x))$$

K[f,p] ist eine Funktion von N nach N, die wir die Zeitkomplexitaet von Programm p fuer f nennen.

Wir interessieren uns lediglich dafuer, mit welcher Groessenordnung die oben definierte Funktion waechst. Deshalb definieren wir : Eine Funktion T : N -> N ist von derselben Ordnung wie die Funktion S(n), wenn Konstanten c1,c2,n1 > Ø existieren, so dass fuer alle n > n1 :

$$c1 * S(n) \leq T(n) \leq c2 * S(n) .$$

Θ(S(n)) definieren wir als die Menge aller Funktionen, welche die Ordnung S(n) haben. Meistens wird als Repraesentant der Ordnung eine "allgemein bekannte" Funktion angegeben, z.B. eine Komposition von Polynomen, Logarithmusfunktionen oder Exponentialfunktionen. Die Ordnung der Zeitkomplexitaet von Programm p nennen wir die asymptotische Zeitkomplexitaet von p. Sie bestimmt unmittelbar die Groesse der Eingabewerte, die vom Programm p mit vernuenftigem Aufwand verarbeitet werden koennen. Allerdings stimmt das nur fuer Eingabewerte, die genuegend gross sind. Ein Programm, dessen Zeitkomplexitaet eine

hohe Wachstumsrate, aber einen kleinen konstanten Koeffizienten hat, ist naemlich fuer kleine Eingabewerte einem Programm ueberlegen, dessen Zeitkomplexitaet eine kleine Wachstumsrate, aber einen grossen konstanten Koeffizienten hat.

Heute werden allgemein die Funktionen, deren asymptotische Zeitkomplexitaet durch ein Polynom beschraenkt ist, als praktisch berechenbar betrachtet. Wir nennen diese Funktionen polynomial berechenbar und bezeichnen die Klasse aller polynomial berechenbaren Funktionen mit P. Sicher sind alle "einfachen" Funktionen in P enthalten. Zudem ist P invariant bei gewissen [PR74] Veraenderungen des Maschinenmodells. Wenn man z.B. die klassische Turingmaschine statt der DZM verwendet (1 Schritt = 1 Zeiteinheit), erhaelt man dieselbe Klasse P.

Im folgenden interessieren wir uns fuer Entscheidungsfunktionen, die auf Worten des Alphabets B={0,1} definiert sind. Jede Teilmenge L von B* nennen wir eine Sprache. Wir sagen, L ist in P, wenn die charakteristische Funktion von L in P ist. Meistens sind die Sprachen, die wir betrachten, gewisse Formelmengen. Die Details der Codierung in (0,1)-Folgen geben wir nicht an.

Wir definieren nun die Klasse NP [CO71]. Die Sprachen in NP erscheinen in vielen Gebieten der Mathematik und sind deshalb besonders interessant (siehe Teil C der Bibliographie). Eine Sprache L ist in NP, wenn es ein nicht-deterministisches Programm p fuer eine DZM mit folgenden Eigenschaften gibt (bei einer nicht-deterministischen Anweisung darf es nur endlich viele Auswahlmoeglichkeiten geben) :

1. p akzeptiert nur Elemente in L, unabhaengig von der Wahl in den nicht-deterministischen Anweisungen.
2. Fuer jedes x in L gibt es eine akzeptierende Berechnung von polynomialer Laenge bezueglich der Laenge von x.

Die folgende Sprache ERFUELLBARKEIT (ERF) ist in NP:
Elemente : Mengen von Klauseln s = {c(1), c(2), ... c(m)} in Variabeln x(1), x(2), ... x(n). Jede Klausel ist eine Menge von Literalen, wobei jeder Literal entweder die Variable x(i) ($1 \leq i \leq n$) oder deren Negation x(i)' ist.
Eigenschaft : Es gibt eine Zuweisung von Wahrheitswerten an die Variabeln, so dass alle Klauseln erfuellt sind. Dabei heisst eine Klausel erfuellt, wenn mindestens einer ihrer Literale den Wahrheitswert wahr hat.

ERF kann auch anders definiert werden : Eine Formel G der Aussagenlogik heisst eine konjunktive Normalform (KNF), gdw. G die Form F(1) & F(2) & ... & F(m) hat, wobei jedes F(i) ($1 \leq i \leq m$) eine Disjunktion (Oder-Operation) von Literalen ist.

Sei G eine Formel der Aussagenlogik, und seien x(1),x(2), ... x(n) die Variabeln, welche in G vorkommen. Eine Interpretation

von G ist eine Zuweisung von Wahrheitswerten an x(1), x(2), ... x(n). Eine Interpretation kann beschrieben werden durch eine Menge I von Literalen, welche fuer jede Variable von G genau einen Literal enthaelt. Wenn eine Variable x in I positiv vorkommt (d.h. x ist in I), dann hat x unter I den Wahrheitswert "wahr". Wenn x negativ in I vorkommt (d.h. x´ ist in I), dann hat x den Wahrheitswert "falsch". Falls eine Formel G wahr ist unter einer Interpretation I, so sagen wir, dass I ein Modell von G ist. Eine Formel, die ein Modell hat, heisst erfuellbar. Also entspricht der Sprache ERF die Menge der erfuellbaren KNF. Eine beliebige aussagenlogische Formel kann durch lineare Vergroesserung in eine aequivalente KNF uebersetzt werden (siehe z.B. [TS68]).

Ein wichtiges Problem ist die Frage, ob P=NP. Vieles weist darauf hin, dass diese Mengen verschieden sind. Davon sind auch die meisten Mathematiker ueberzeugt. Trotzdem bin ich aufgrund der Resultate dieser Arbeit noch nicht ganz ueberzeugt von der Vermutung P≠NP.

Die Frage, ob P=NP, laesst sich reduzieren auf die Frage, ob die Sprache ERF in P ist [CO71]. Seien L und M Sprachen. Dann heisst L polynomial reduzierbar auf M (L<M), wenn es eine Funktion f in P von B* nach B* gibt, so dass f(x) in M ist, gdw. x in L ist. Wenn L<M und M in P ist, so ist auch L in P. Diese Reduzierbarkeit nennt man mehrdeutige Reduzierbarkeit.

Eine Sprache L heisst in polynomialer Zeit Turing-reduzierbar zu einer Sprache M, wenn es eine Orakel-Turingmaschine TM und ein Polynom p gibt, so dass x in L ist, gdw. TM das Element x mit M als Orakel in p(Laenge(x)) Schritten akzeptiert. Interessante Eigenschaften dieser und anderer polynomialer Reduzierbarkeiten finden sich in [LA74,LA75] (siehe auch Bibliographie Teil F).

Sei D eine Menge von Sprachen. Eine Sprache ds in D heisst D-schwierig, wenn fuer alle Sprachen d in D gilt : d<ds. Fuer eine Sprache ds, die D-schwierig ist, gilt : Wenn ds in P ist, so ist die ganze Menge D in P. Wenn hingegen fuer eine D-schwierige Sprache ds bewiesen ist, dass sie nicht in P ist, so gilt dies nicht fuer ganz D. Deshalb brauchen wir folgende Definition : Eine Sprache dv in D heisst D-vollstaendig, wenn dv D-schwierig ist und es eine Sprache d in D gibt mit dv<d. Wenn eine D-vollstaendige Sprache nicht in P ist, so ist P verschieden von D. Denn, wenn jede Sprache d in D in P waere, so waere auch jede D-vollstaendige Sprache in P.

Sei nun D=NP. Cook [CO71] hat bewiesen, dass L in NP, gdw. L < ERF. Deshalb ist ERF NP-schwierig. ERF ist auch NP-vollstaendig, weil ERF in NP liegt. Mit ERF(k) bezeichnen wir die Teilsprache von ERF, fuer die gilt: Jede Klausel einer KNF in ERF(k) hat hoechstens Laenge k, d.h. sie enthaelt hoechstens k Literale. ERF(k) ist NP-vollstaendig fuer k>2. Hingegen ist

ERF(2) in P.

Wenn P=NP, so ist NP abgeschlossen unter Komplementbildung, denn P ist abgeschlossen unter Komplementbildung. Die Frage, ob NP unter Komplementbildung abgeschlossen ist, steht in direktem Zusammenhang mit der Untersuchung von Beweissystemen [CO74]. Sei L eine Sprache ueber B={0,1}. Ein Beweissystem fuer L ist ein nicht-deterministischer Algorithmus BW(L) mit Eingabewerten x in B*. Falls es fuer ein x eine Berechnung w gibt, die x akzeptiert, so ist w ein Beweis, dass x in L ist. Wir setzen voraus, dass BW(L) korrekt ist, d.h. nur Elemente x in L akzeptiert. Weiter verlangen wir, dass es deterministisch in polynomialer Zeit bezueglich der Laenge von w moeglich ist, zu ueberpruefen, ob eine Berechnung w entsprechend den Vorschriften des Algorithmus BW(L) ausgefuehrt wurde. Beweissysteme koennen auch aufgefasst werden als Schlussregelsysteme.

Sei L eine Sprache und BW(L) ein Beweissystem fuer L. Sei c(n) die Menge saemtlicher Elemente x in L, welche feste Laenge n haben. s(x) sei die Laenge des kuerzesten Beweises fuer x im Beweissystem BW[L]. Wir nennen

$$K[BW(L)]\ (n) = \max\ [x \text{ in } c(n)]\ s(x)$$

die Komplexitaet des Beweissystems BW(L). Die asymptotische Komplexitaet eines Beweissystems definieren wir wie bei Programmen.

Ein Beweissystem BW(L) heisst polynomial, wenn die Laenge des kuerzesten Beweises fuer alle x in L beschraenkt ist durch ein Polynom in der Laenge von x, d.h. wenn die asymptotische Komplexitaet durch ein Polynom beschraenkt ist. Wenn eine Sprache ein polynomiales Beweissystem hat, so ist sie in NP.

Sei UNERF die Sprache der unerfuellbaren KNF. Dann gilt : NP ist abgeschlossen unter Komplementbildung, gdw. UNERF ein polynomiales Beweissystem hat [CO74].

Ein bekanntes Beweissystem fuer UNERF ist Resolution. Zwei Klauseln c1 und c2 stossen zusammen, wenn es genau einen Literal in c1 gibt, der in c2 komplementiert vorkommt. Falls die Klauseln d1 = c1 + {x} und d2 = c2 + {x'} zusammenstossen, dann ist r = c1 + c2 die Resolvente von d1 und d2. Wir sagen, r wurde erhalten durch Anwendung von Resolution. d1 und d2 heissen Elternklauseln, x heisst aufgeloeste Variable.

Ein Resolutionsbeweis fuer die Unerfuellbarkeit einer KNF s ist eine Folge von Klauseln c(1), c(2), ... c(k), so dass c(k) die leere Klausel ist. Fuer $1 \leq i \leq k-1$ muss gelten, dass c(i+1) eine Resolvente von Klauseln in s + {c(1), c(2), ... c(i)} ist. Man kann zeigen, dass eine KNF s unerfuellbar ist, gdw. ein Resolutionsbeweis existiert. Solche Beweissysteme nennt man vollstaendig. Regulaere Resolution, eine einschraen-

kende Art von Resolution, ist auch vollstaendig. In [TS68, GA74] wird gezeigt, dass dieses Beweissystem nicht polynomial ist. Fuer Resolution wird ebenfalls vermutet, dass sie nicht polynomial ist [COO76].

Ein Beweissystem BW1(L1) fuer L1 heisst polynomial reduzierbar auf ein Beweissystem BW2(L2) fuer L2 (BW1(L1) < BW2(L2)), wenn
1. L1 < L2 (Die Uebersetzungsfunktion sei f.)
2. fuer jeden Beweis w1(x1) fuer ein x1 in L1 ein Beweis w2(f(x1)) fuer f(x1) in L2 existiert, so dass die Laenge von w2(f(x1)) beschraenkt ist durch ein Polynom in der Laenge von w1(x1).

Zwei Beweissysteme BW1(L1) und BW(L2) heissen aequivalent, wenn BW1(L1) < BW2(L2) und BW2(L2) < BW1(L1) .

In [CO74] werden verschiedene Beweissysteme fuer UNERF in bezug auf ihre gegenseitige Reduzierbarkeit untersucht. Als ein starkes Beweissystem erweist sich erweiterte Resolution.

Bei der erweiterten Resolution koennen Hilfsvariabeln eingefuehrt werden. Sei s eine KNF. Fuer zwei Literale x,y in s und eine neue Hilfsvariable h sei

$$A(h,x,y) = \{(x',h) , (y',h) , (x,y,h')\} .$$

Man beachte, dass die Menge der Klauseln A(h,x,y) aequivalent ist zu h = x v y . Die Erweiterungsregel ist folgendermassen definiert:
1. Waehle x und y in s.
2. Erweitere s um die Klauseln A(h,x,y).

Auch von diesem Beweissystem wird in [CO75] vermutet, dass es nicht polynomial ist.

Als Sprache zur Beschreibung von Algorithmen verwende ich Pidgin-Algol, wie es in [AHO75] beschrieben ist. Kommentare werden zwischen (* und *) eingeschlossen, z.B. : Sei s eine KNF und l ein Literal in s. Die Prozedur REDUZIERE(s,l) leiste folgendes : Alle Klauseln, die l enthalten, werden gestrichen. In allen Klauseln, welche l' enthalten, wird l' gestrichen.

Beschreibung in Pidgin-Algol:

```
procedure REDUZIERE(s,l);
(* s ist eine KNF und l ein Literal von s *)
begin
   for jede Klausel c, die l enthaelt do
   entferne c aus s;
   for jede Klausel c, die l' enthaelt do
   entferne den Literal l' aus c
end
```

Wenn ein Parameter in einem Prozeduraufruf irrelevant ist, wird er durch * ersetzt.

1. Superresolution

Superresolution ist eine effiziente Konkretisierung der in [RO68] eingefuehrten verallgemeinerten Resolution. Sie ordnet jeder misslungenen Modellkonstruktion eine verallgemeinerte Resolvente zu. Superresolution ist nicht nur ein weiteres der zahlreichen vollstaendigen Resolutionsbeweissysteme, sondern in verschiedener Hinsicht beweisbar besser als Resolution und deren Abarten.

Auf dem Gebiete des automatischen Beweisens werden verschiedene einschraenkende Varianten von Resolution untersucht, die aber immer noch vollstaendig sind (Siehe Bibliographie Teil A). Diese Untersuchungen werden durch folgende Erfahrung motiviert: Automatische Beweisprogramme haben die unangenehme Eigenschaft, enorm viele Folgerungen zu ziehen, die fuer den gesuchten Beweis irrelevant sind (siehe z.B. [MAR75]). Wenn man eingeschraenkte Schlussregeln verwendet, ist diese Gefahr vermindert. Allerdings verlaengern solche eingeschraenkten Schlussregeln meistens die Beweise [KI72, ME71, SHO76]. Es ist intuitiv einleuchtend, dass mit eingeschraenkten Schlussregeln im allgemeinen laengere Beweise entstehen. Fuer Superresolution kann ich hingegen beweisen:

1. Superresolution ist eine starke Einschraenkung von Resolution (siehe Abschnitt 1.5) und
2. Superresolutionsbeweise sind kuerzer als Resolutionsbeweise (siehe Abschnitt 1.6).

Deshalb eignet sich Superresolution (mehr als alle anderen bekannten Abarten von Resolution) vortrefflich als Schlussregel in einem automatischen Beweisprogramm.

1.1 Das vollstaendige Beweissystem Superresolution

Zur Definition des Begriffs Superresolvente benoetigen wir die im folgenden beschriebene Prozedur SETZE.

SETZE findet fuer eine KNF s und einen Literal l diejenigen Literale, die wahr gesetzt werden muessen, wenn l wahr gesetzt wird. Ein Literal l1 heisst durch l eindeutig bestimmt, falls eine unerfuellte Klausel entsteht, wenn l wahr und l1 falsch gesetzt wird. Die Boolesche Variable WIDERSPRUCH wird wahr, wenn die eindeutig bestimmten Literale eine Klausel von s unerfuellt lassen.

```
procedure SETZE(l,s,WIDERSPRUCH);
begin
   el := {l} ; WIDERSPRUCH := false ;
   repeat
      waehle ein Element ll aus el ;
      el := el - {ll} ;
      seien kpll die Klauseln in s,
      welche den Literal ll enthalten ;
      for jede Klausel c in kpll do entferne c aus s;
      seien knll die Klauseln in s, welche den Literal ll´
      enthalten;
      for jede KLausel c in knll do
      begin
         streiche den Literal ll´ in der KLausel c von s;
         if c enthaelt genau einen Literal l2 then
         (* l2 heisst eindeutig bestimmt *)
         if l2´ in el then WIDERSPRUCH := true else
         el := el + {l2}
      end
   until el={} ;
end;
```

Informell ist eine Superresolvente sr einer KNF s eine Klausel, fuer die folgendes gilt: Seien alle Literale in sr mit Hilfe von SETZE so gesetzt, dass sr unerfuellt ist. Die Variablen, die zu anderen Literalen als denen in sr gehoeren, seien noch nicht interpretiert. Dann darf keine Klausel in s unerfuellt sein. Jetzt muss es eine Variable v geben, deren beide Interpretationen durch Konsequenzenbildung (d.h. durch Beruecksichtigen der eindeutig bestimmten Literale) eine unerfuellte Klausel entstehen lassen. Die Variable v nennen wir lernend.

Nun geben wir die exakte Definition fuer den allgemeineren Begriff Superresolvente k-ten Grades.

Definition 1.1.1 : Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente k-ten Grades (k>1) einer KNF s, wenn man SETZE(l(i),s,w) der Reihe nach fuer i = 1, 2, ... n ausfuehren kann und immer w=false erhaelt, und wenn dann

1. k Variablen v(1), v(2), ... v(k) existieren, so dass fuer alle Folgen vv(1), vv(2), ... vv(k) (vv(j) in {v(j),v(j)´} (1<j<k)) die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k ausgefuehrt, immer w=true liefert und
2. fuer jedes k1<k eine Folge vv(1), vv(2), ... vv(k1) existiert, so dass die Prozedur SETZE(vv(i),s,w), der Reihe nach fuer i = 1, 2, ... k1 ausgefuehrt, w=false liefert.

Eine Klausel {l(1), l(2), ... l(n)} heisst Superresolvente einer KNF s, wenn man SETZE(l(i),s,w) der Reihe

nach fuer i = 1, 2, ... n ausfuehren kann und immer w=false erhaelt, und wenn dann eine Variable v existiert, fuer welche sowohl bei SETZE(v,s,w) als auch bei SETZE(v´,s,w) die boolesche Variable w=true wird.

Bemerkungen:
1. Man beachte, dass eine Superresolvente ersten Grades eine Superresolvente ist.
2. Bei der Konstruktion einer Superresolventen sr einer KNF s werden viele Klauseln von s, manchmal sogar alle (wie im folgenden Beispiel), beruecksichtigt. Eine Superresolvente ist also ein "Kondensat" von vielen Klauseln, eine Resolvente hingegen nur von 2.
3. Fuer eine Resolvente ist es einfach zu ueberpruefen, dass sie bei jedem existierenden Modell erfuellt ist. Man hat dazu nur die beiden Elternklauseln zu finden. Eine Superresolvente ist auch bei jedem existierenden Modell erfuellt, d.h. eine Superresolvente ist wie eine Resolvente eine Folgerung. Diese Eigenschaft ist zwar verborgen, doch durch iterierte Anwendung von SETZE einzusehen.

Beispiel:
1. a b
2. c d
3. e f
4. a´ c´
5. a´ e´
6. c´ e´
7. b´ d´
8. b´ f´
9. d´ f´

Die leere Klausel ist eine Superresolvente dieser KNF, da jede Variable lernend ist.

Fuer diejenigen Leser, denen die Programmiersprachterminologie nicht vertraut ist, folgt nun eine Definition des Begriffs Superresolvente in der ueblichen mathematischen Terminologie. Sei s eine beliebige aussagenlogische Formel und LIT(s) die Menge der Literale in s. Eine Teilinterpretation von s ist eine Teilmenge einer Interpretation aller Variablen von s. Ein Literal l in LIT(s) heisst durch eine Teilinterpretation I einfach bestimmt, wenn s unter der Interpretation I+{l´} unerfuellt ist oder wenn l in I ist (wir schreiben (s,I)->l). Eine Formel s heisst unerfuellt unter der partiellen Interpretation I, wenn nach der Elimination der Konstanten, die durch I in s eingefuehrt werden, "falsch" erhalten wird. Ein Literal l in LIT(s) heisst durch eine Teilinterpretation I bestimmt in der Formel s, wenn es Literale g(1), g(2), ... g(n)=l (n>0) in LIT(s) gibt, so dass (s, I+{g(1), g(2), ... g(i-1)})->g(i) fuer 1<i<n (wir schreiben (s,I)=>l).
Sei D(s,I) die Menge aller Literale von s, die durch I be-

stimmt sind, d.h.

$$D(s,I) = \{l \text{ in } LIT(s) : (s,I) => l\}.$$

D(s,I) kann komplementaere Paare von Literalen enthalten, d.h. eine Variable v und ihr Komplement v´.

Definition 1.1.1A : Sei s eine aussagenlogische Formel und c eine Klausel mit Literalen in s. Sei I die Menge der komplementierten Literale in c. c heisst eine Superresolvente von s, wenn
1. D(s,I) kein komplementaeres Literalpaar enthaelt und wenn
2. eine Variable v existiert, so dass sowohl D(s,I+{v}) als auch D(s,I+{v´}) ein komplementiertes Literalpaar enthaelt.

Lemma 1.1.1 : Sei s eine erfuellbare KNF und M ein Modell von s. Dann ist jede Superresolvente beliebigen Grades von M erfuellt.

Beweis:
Sei sr eine Superresolvente k-ten Grades von s. Annahme: sr sei unerfuellt beim Modell M. Wegen der Definition 1.1.1 gibt es k Variablen v(1),v(2), ... ,v(k) in s - sr, so dass bei allen Interpretationen dieser Variablen mit Hilfe von SETZE eine unerfuellte Klausel in s entsteht. Also kann eine Interpretation, welche sr nicht erfuellt, sicher nicht zu einem Modell erweitert werden. Widerspruch. #

Definition 1.1.2 : Ein Superresolutionsbeweis k-ten Grades fuer eine KNF s ist eine Folge von Klauseln c(1),c(2), ... c(m), so dass c(m)={} und fuer i zwischen 1 und m-1 die Klausel c(i+1) eine Superresolvente hoechstens k-ten Grades der KNF s + {c(1), c(2), ... c(i)} ist.
Einen Superresolutionsbeweis ersten Grades nennen wir einen Superresolutionsbeweis.

Bemerkung:
Automatischer Gleichheits- und Teilklauseltest: Man beachte, dass gemaess Definition des Begriffs "Superresolvente" nicht zweimal dieselbe Superresolvente oder eine Superresolvente, die eine frueher erzeugte enthaelt, in einem Superresolutionsbeweis vorkommen kann. Dies ist bei Resolutionsbeweisen nicht garantiert, weshalb ein aufwendiger Test noetig ist.

Lemma 1.1.2 : Fuer jedes k gilt : Eine KNF s ist unerfuell-

bar genau dann, wenn ein Superresolutionsbeweis k-ten Grades existiert.

Beweis:
Wenn ein Superresolutionsbeweis k-ten Grades fuer s existiert, so ist s nach Lemma 1.1.1 unerfuellbar. Die Umkehrung wird aus dem folgenden klar. #

Bemerkung :
Fuer eine KNF mit n Variablen ist Superresolution log(n)-ten Grades immer noch ein Beweissystem, d.h. das Ueberpruefen des Beweises ist polynomial moeglich. k braucht also nicht konstant zu sein. Superresolution ersten Grades und Superresolution log(n)-ten Grades sind trotzdem polynomial aequivalente Beweissysteme, denn jede Superresolvente log(n)-ten Grades kann durch Superresolventen linear simuliert werden.

Im naechsten Kapitel beschreiben wir einen universellen Algorithmus SR, der jede interessante Superresolvente erzeugen kann. Die Existenz und die Eigenschaften dieses Algorithmus werden uns behilflich sein, verschiedene Einsichten in Superresolution zu gewinnen, insbesondere wie man "kurze" Superresolventen finden kann. Eine wichtige Eigenschaft von SR ist, dass sich ein beliebiger Superresolutionsbeweis auf einen von SR erzeugten verkuerzen laesst. Deshalb haben die von SR erzeugten Superresolutionsbeweise eine zentrale Stellung. Algorithmus SR wird in dieser Arbeit haeufig verwendet, und deshalb ist eine genaue Kenntnis von ihm unerlaesslich fuer das Verstaendnis der spaeteren Abschnitte.

1.2 Beschreibung von Algorithmus SR

SR ist ein Algorithmus mit nicht-deterministischen Anweisungen, der auf KNF operiert. In jeder nicht-deterministischen Anweisung hat er eine endliche Auswahl von Alternativen. Wenn nicht ausdruecklich erwaehnt, gelten die Aussagen ueber SR fuer jede beliebige Wahl in den nicht-deterministischen Anweisungen. SR kann deshalb als das erzeugende Element einer ganzen Algorithmenklasse aufgefasst werden.

Auf erfuellbaren KNF berechnet SR ein Modell und auf unerfuellbaren KNF einen Superresolutionsbeweis, unabhaengig von der Variante, welche bei nicht-deterministischen Anweisungen gewaehlt wird. Diese Wahl hat jedoch bei erfuellbaren KNF einen wesentlichen Einfluss auf die Berechnungszeit von SR, weil die Interpretation von Variabeln im wesentlichen auch ueber nicht-deterministische Anweisungen geschieht. Waehlt SR naemlich ein Modell, so terminiert er sofort.

Sei s eine Eingabe-KNF. SR versucht fuer s ein Modell zu konstruieren, indem SR nach bestimmten Regeln eine Interpretation konstruiert. Einer Interpretationskonstruktion von SR entspricht ein Schritt von SR. Falls SR im laufenden Schritt kein Modell findet, wird eine Superresolvente r in s aufgenommen ("gelernt"). r hat die Aufgabe, zu verhindern, dass bei einer spaeteren Interpretationskonstruktion nochmals der "Fehler" (nur bei erfuellbaren KNF ist es ein eigentlicher Fehler), der beim letzten Schritt begangen wurde, wiederholt wird. Falls r leer ist, hat man Unerfuellbarkeit bewiesen. Sonst wird ein weiterer Schritt ausgefuehrt. Fuer die Konstruktion von r verwendet SR eine Menge von Baeumen, die bei der Interpretationskonstruktion aufgebaut werden.

Ist I eine von SR bereits konstruierte Teilinterpretation von s, die erweitert wird, und l ein Literal, der als naechster in I aufgenommen werden koennte, dann muss genau einer der drei Faelle eintreten:

1. Fall "waehle"
   Kein Literal im noch vorhandenen s ist durch die Anwendung von SETZE eindeutig bestimmt. Auch l´ koennte anstelle von l in I aufgenommen werden, ohne dass durch SETZE ein Widerspruch entsteht. Wir nennen die Variable, die zu l gehoert, waehlbar.
2. Fall "unerfuellt"
   Sowohl wenn l, als auch wenn l´ in I aufgenommen wuerde, entstuende ein Widerspruch. Deshalb kann I nicht zu einem Modell erweitert werden. Es wird eine Superresolvente sr gebildet, die eine Teilmenge der komplementierten Literale von I ist. Die Variable, die zu l gehoert, ist die lernende Variable der Superresolventen sr.
3. Fall "isoliert"
   Wenn l´ in I aufgenommen wuerde, so wuerde ein Widerspruch entstehen. Wir nennen die Variable, die zu l gehoert, isoliert.

Zuerst beschreibe ich Algorithmus SR in der ueblichen mathematischen Terminologie. Anschliessend folgt eine Beschreibung in Pidgin-Algol [AHO75].

Um das Wesentliche herausheben zu koennen, beschreibe ich zuerst einen etwas einfacheren Algorithmus SR´. Mit Hilfe von SR´ kann SR einfach erklaert werden.

Sei s die Eingabe-KNF. In einem Schritt von SR´ wird nach folgenden Regeln eine Interpretation konstruiert und, wenn kein Modell gefunden wird, eine Superresolvente gebildet:

1. Die folgenden Operationen werden auf einer Kopie von s durchgefuehrt. Das momentane s wird nach diesem Schritt wieder verwendet. Es wird eine beliebige Variable v von s gewaehlt und festgestellt, welche der Eigenschaften "waehlbar", "lernend" und "isoliert" gilt. (Es muss genau eine gelten.) Meistens wird v waehlbar sein.

a) waehlbar
Es wird ein Literal l von v gewaehlt. Dann wird l in die Menge GEWAEHLT aufgenommen.
b) isoliert
Sei l ein Literal von v, so dass kein Widerspruch entsteht, wenn l wahr gesetzt wird.
c) lernend
Es wird ein Literal l von v gewaehlt und eine Superresolvente sr gebildet, die in eine Menge sres aufgenommen wird. sr enthaelt gewisse komplementierte Literale der momentanen Menge GEWAEHLT. Diese Teilmenge garantiert bereits, dass durch Konsequenzenbildung nachgeprueft werden kann, dass die entsprechende Teilinterpretation (die komplementierten Literale der Teilmenge) nicht zu einem Modell erweitert werden kann.

Man beachte, dass die Menge G der komplementierten Literale von GEWAEHLT auch eine Superresolvente ist. Die Eigenschaften von sr sind in Abschnitt 1.3 etwas genauer analysiert.

2. Der Literal l wird nun mit SETZE wahr gesetzt, d.h. l und die durch l eindeutig bestimmten Literale werden wahr gesetzt und die betroffenen Klauseln entweder gestrichen oder gekuerzt. Dabei wird die Geschichte der Konsequenzenbildung festgehalten, indem fuer jeden Literal l1, der durch l eindeutig bestimmt ist, die Menge der durch l1 eindeutig bestimmten Literale gebildet wird. Diese Mengen werden bei der Bestimmung der Superresolventen verwendet.

3. Falls s noch nicht leer ist, werden die Punkte 1. und 2. wiederholt, bis s leer ist.

4. s wird fuer den naechsten Schritt vorbereitet, falls kein Modell oder die leere Superresolvente erhalten wurde: s erhaelt wieder den Wert vor dem letzten Schritt, ergaenzt um eine aus sres gewaehlte Superresolvente. Anschliessend werden allenfalls Vereinfachungen durchgefuehrt, z.B. Klauseln der Laenge 1 eliminiert.

Im naechsten Schritt wird analog mit der erweiterten KNF eine Interpretationskonstruktion durchgefuehrt. Bei zwei verschiedenen Schritten kann nicht dieselbe Superresolvente gebildet werden. Deshalb bricht der Algorithmus nach endlich vielen Schritten ab, weil nur endlich viele verschiedene Superresolventen existieren. Ueber die Anzahl der Superresolventen kann ich nur relativ zur Anzahl der Resolventen etwas Interessantes aussagen (Siehe Abschnitt 1.5). Es gibt wesentlich weniger Superresolventen als Resolventen.

Soweit die Beschreibung von SR'. Algorithmus SR unterscheidet sich folgendermassen von SR': Bevor eine Variable v gewaehlt wird, evaluiert SR fuer jede Variable von s, ob sie waehlbar,

lernend oder isoliert ist. Wenn es eine lernende resp. isolierte Variable gibt, wird eine solche gewaehlt und sonst eine waehlbare. Ausser dieser Wahlvorschrift ist SR genau gleich wie SR'.

Nun folgt die Beschreibung in Pidgin-Algol. Der Algorithmus SR benuetzt vier separat definierte Prozeduren, die auf den folgenden Seiten beschrieben sind. Danach wird die Arbeitsweise anhand eines Beispiels erlaeutert.

Globale Daten von SR:

GEWAEHLT, I1: sind Mengen von Literalen, die als Interpretationen der Variabeln in s aufzufassen sind. GEWAEHLT ist eine Teilmenge von I1.

I,I1: sind Array's von Literalmengen, die als Interpretationen der Variabeln in s aufzufassen sind.

LERNEND: ist ein array von Booleschen Variabeln. Die Indizes sind die Variabeln in s. LERNEND[v] wird wahr, wenn die Interpretation I1 sich nicht zu einem Modell erweitern laesst.

s: ist eine KNF, zu Beginn die Eingabe-KNF.

SCHRITT: zaehlt, wieviele Interpretationskonstruktionen durchgefuehrt werden.

UEBRIG: ist eine KNF, die aus s entsteht durch Entfernen von Klauseln und Markieren von Literalen.

UNERFUELLBAR: Diese Boolesche Variable wird wahr, wenn die Eingabe-KNF unerfuellbar ist.

ISOLIERT: ist ein array von Booleschen Variabeln. Die Indizes sind die Variabeln in s. ISOLIERT[v] wird wahr, wenn die Interpretation von v eindeutig bestimmt ist mit Hilfe der Prozedur SETZE.

WALD: ist ein array von Baeumen, in dem die Information ueber gewisse Klauseln, die aus UEBRIG gestrichen wurden, gespeichert ist. Die Punkte eines Baumes b sind Literale. Den Kanten von b ist als zusaetzliche Information eine Menge von Literalen zugeordnet. Die Indizes von WALD sind die Literale von s. Die Wurzel von WALD[l] ist der Literal l.

wvar: ist ein array von Variablenmengen. Indizes sind die Literale in s.

Die Prozedur VEREINFACHE(s)

Globale Variable : UNERFUELLBAR

VEREINFACHE fuehrt triviale Verkuerzungen der KNF s durch, naemlich:

1. Elimination von Klauseln der Laenge 1.
2. Anwendung von Subsumption (wenn eine Klausel d in einer Klausel c enthalten ist, so wird nur d behalten).
3. Wenn eine Variable v nur positiv vorkommt in s (nur v kommt in s vor), so koennen alle Klauseln, die v enthalten, gestrichen werden. Analoges gilt, wenn v nur negativ vorkommt.
4. Wenn l1=l2 gilt, wird l1 ueberall durch l2 ersetzt.

Beschreibung in Pidgin-Algol:

```
procedure VEREINFACHE(s);
   begin
      AENDERUNG := false ;
      repeat
         while s enthaelt einen Literal l in einer Klausel der
         Laenge 1 do
         begin
            REDUZIERE(s,l) ;
            UNERFUELLBAR wird wahr, wenn eine Klausel der
            Laenge Ø entstanden ist ;
            AENDERUNG := true ;
         end;
         while s enthaelt eine Klausel c, die eine andere
         Klausel enthaelt do
         begin
            eliminiere c in s ;
            AENDERUNG := true ;
         end;
         while s enthaelt einen Literal l, ohne dass l´ in s
         vorkommt do
         begin
            REDUZIERE(s,l) ;
            AENDERUNG := true ;
         end;
         while es gibt Klauselpaare {l1,l2´},{l1´,l2} in s do
         begin
            ersetze in allen Klauseln von s den Literal l1
            durch l2 ;
            AENDERUNG := true ;
         end;
      until not AENDERUNG ;
   end;
```

Bemerkung : VEREINFACHE laesst sich geschickter formulieren, weil nicht jede while-Schleife die Bedingungen in allen anderen while-Anweisungen veraendert.

Die Prozedur SETZEB(l,UEBRIG,WIDERSPRUCH)

Globale Variablen: I[l], WALD[l], wvar[l]

Im Unterschied zu SETZE speichert SETZEB seine Geschichte in der Datenstruktur WALD. Diese Information wird spaeter dazu verwendet, die Superresolvente zu konstruieren.
Der Literal l wird wahr gesetzt, und alle dadurch eindeutig bestimmten Literale werden auch wahr gesetzt und in I[l] gespeichert. Wenn ein eindeutig bestimmter Literal gefunden wird, erweitert SETZEB den Baum WALD[l] durch eine Kante. Als Label dieser Kante werden Literale derjenigen Klausel eingetragen, welche den eindeutig bestimmten Literal verursachte. Die Variable WIDERSPRUCH wird wahr, wenn beim Setzen der eindeutig bestimmten Literale eine unerfuellte Klausel entsteht. Wenn WIDERSPRUCH wahr ist, so ist wvar[l] die Menge von Variabeln, die sowohl wahr wie auch falsch bestimmt sind.

Beschreibung in Pidgin-Algol:

```
procedure SETZEB(l,UEBRIG,WIDERSPRUCH);
   begin
      el := {l} ;
      WIDERSPRUCH := false; wvar[l] := {};
      initialisiere WALD[l] durch seine Wurzel l;
      repeat
         (* Nicht-deterministische Wahl [el] *)
         sei l1 ein Element aus el; el := el - {l1} ;
         I[l] := I[l] + {l1} ; seien kpl1 die Klauseln in UEB-
         RIG, welche den Literal l1 enthalten;
         for jede Klausel c in kpl1 do entferne c aus UEBRIG;
         seien knl1 die Klauseln in UEBRIG, welche den Literal
         l1´ enthalten ;
         for jede Klausel c in knl1 do
         begin
            markiere den Literal l1´ in der Klausel c von UEB-
            RIG ;
            if c enthaelt genau einen Literal l2, der nicht
            markiert ist then
            begin
               waehle einen Punkt l1 in WALD[l], dem der Lite-
               ral l1 zugeordnet ist;
               erweitere WALD[l] durch eine Kante ka von l1
               nach dem neuen Punkt l2;
               label[ka] := c - {l1} - {l2} ;
               if l2´ in el then
               begin
                  wvar[l] := wvar[l] + { Variable, die zu Li-
                  teral l2 gehoert } ;
                  WIDERSPRUCH := true ;
               end else el := el + {l2};
            end;
         end;
      until el={} ;
   end;
```

Die Prozedur LABELSAMMELN(l,vp,vn,w)

Globale Variable: WALD[l]

Nach der Ausfuehrung enthaelt die Menge w die Literale, welche als Label der Kanten auf dem Weg von der Wurzel von WALD[l] nach den Literalen vp resp. vn auftreten.

Beschreibung in Pidgin-Algol

```
procedure LABELSAMMELN(l,vp,vn,w);
   begin
      verwende WALD[l] und bestimme die Menge w der Labels auf
```

```
      den Wegen von l nach vp und nach vn ;
   end;
```

Die Prozedur AUFROLLEN(w)

Globale Variabeln: GEWAEHLT, WALD

Vor der Ausfuehrung von AUFROLLEN enthaelt die Menge w eventuell einen Literal ng, der durch andere Literale eindeutig bestimmt wurde, d.h. es gibt einen Literal g, so dass WALD[g] den Literal ng´ an einem Punkt vng enthaelt. ng´ ist eindeutig bestimmt, weil g in I1 und die komplementierten Literale der Label der Kanten auf dem Weg von g nach ng´ in I1 sind. Deshalb wird ng in w ersetzt durch g´ + {Label der Kanten auf dem Weg von g nach vng im WALD[g]}. Diese Operation wird solange wiederholt, bis w nur noch Elemente enthaelt, die in der elementweise komplementierten Menge GEWAEHLT sind.

Beschreibung in Pidgin-Algol

```
procedure AUFROLLEN;
   begin
      while w enthaelt Literale, die durch andere bestimmt
      wurden do
      begin
         (* nicht-deterministische Wahl [w] *)
         sei ng ein Literal in w, der in SETZEB durch einen
         anderen Literal g bestimmt wurde ; d.h. WALD[g] enthaelt den Literal ng´ an den Punkten ; waehle ein
         Vorkommen vng von ng´ an den Punkten von WALD[g]; ersetze ng in w durch g´ zusammen mit den Labels des
         Weges von g nach vng in WALD[g] ;
         w := w - {ng} ;
      end
      (* w ist eine Teilmenge von GEWAEHLT *)
   end
```

Algorithmus SR

Die Grobstruktur von SR

```
Eingabe von s;
VEREINFACHE(s);
while Entscheid nicht gefaellt do
begin
   (* naechster Schritt: *)
   UEBRIG := s ;
   while in UEBRIG gibt es Variabeln zu interpretieren do
   begin
      Evaluiere die Variabeln in UEBRIG mit der for-Schleife
```

```
      EVAL. Die Evaluation kann 3 verschiedene Ergebnisse
      liefern : 1.Fall : "waehle", 2.Fall : "unerfuellt" und
      3.Fall : "isoliert". In jedem Fall wird ein Literal l
      gewaehlt, der anschliessend mit SETZEB(l,UEBRIG,*) in-
      terpretiert wird.
   end;
   (* Falls kein Modell gefunden wurde, so wurde s um eine Su-
   perresolvente erweitert. *)
   VEREINFACHE(s);
end;
Ausgabe der Entscheidung. (* Die Entscheidung lautet "erfuell-
bar", falls ein Modell gefunden wurde und "unerfuellbar",
falls die leere Superresolvente gebildet wurde *)

Nun folgt die detaillierte Beschreibung von SR:

begin
   UNERFUELLBAR := false ;
   SCHRITT := Ø ; VEREINFACHE(s) ;
   while nicht alle Klauseln von s sind erfuellt and not UNER-
   FUELLBAR do
   begin
      SCHRITT := SCHRITT + 1 ;
      UEBRIG := s ; GEWAEHLT := {} ; I1 := {} ;
      V1 :=  {die Menge der Variabeln in UEBRIG, die
      noch nicht definitiv interpretiert sind} ;
      while V1 # {} and
         in UEBRIG sind nicht alle Klauseln gestrichen
            do (* while-Schleife V1 :  *)
      begin
         for jede Variable v in V1 do  (* for-Schleife EVAL:*)
         begin
            UEBRIG1 := UEBRIG ; UEBRIG2 := UEBRIG ;
            SETZEB(v,UEBRIG1,WID1);
            SETZEB(v',UEBRIG2,WID2);
            if WID1 and not WID2 then
            begin
               ISOLIERT[v] :=  true ;
            end else
            if not WID1 and WID2 then
            begin
               ISOLIERT[v] := true ;
            end else
            if WID1 and WID2 then
            begin
               LERNEND[v] := true;
               (* Die Variable v nennen wir lernend *)
            end else WAEHLBAR[v] := true;
            if LERNEND[v] then
            begin
               (* nicht-deterministische Wahl [wvar] *)
               waehle eine Variable wv in wvar[v] und waehle
               ein Vorkommen
```

```
        vpwv von wv und ein Vorkommen vnwv
        von wv´ an den Punkten von WALD[v]
        begin
           LABELSAMMELN(WALD[v],vpwv,vnwv,w1);
           (* nicht-deterministische Wahl [wvar] *)
           waehle eine Variable wv1 in wvar[v´]
           und waehle ein Vorkommen vpwv1 von wv1
           und ein Vorkommen vnwv1 von wv1´ an den
           Punkten von WALD[v´] ;
           begin
              LABELSAMMELN(WALD[v´],vpwv1,vnwv1,w2);
              AUFROLLEN(w1+w2);
              UNERFUELLBAR := w=[] ;
              res[v] := w ;
           end;
        end;
     end
end; (* Ende der for-Schleife EVAL *)
if fuer alle Variabeln v in V1 gilt WAEHLBAR[v] =
true then
(* Fall "waehle" :  *)
begin
     if es gibt Variabeln in V1, die nicht nur
     positiv oder nur negativ in UEBRIG vorkommen then
     begin
        waehle einen Literal l einer Variabeln in V1,
        der nicht nur positiv oder nur
        negativ in UEBRIG vorkommt;
        GEWAEHLT := GEWAEHLT + {l} ;
     end else waehle einen Literal l einer Variabeln in
     V1;
end else
if es gibt lernende Variabeln in V1 then
(* Fall "unerfuellt" :  *)
begin
     sei resver die Vereinigungsmenge der Klauseln in
     res[v] fuer alle lernenden Variabeln v;
     konstruiere eine neue Menge sres aus resver, indem
     in resver
     alle Klauseln entfernt werden, die in andern
     Klauseln von resver enthalten sind.
     (* nicht-deterministische Wahl [sres] *)
     waehle eine Klausel sr aus sres;
     s := s + {sr} ;
     (* s wird um die Superresolvente sr erweitert *)
     sei v1 eine lernende Variable mit der
     sr erhalten wurde;
     waehle einen Literal l von v1;
     entferne alle Klauseln von UEBRIG, die unerfuellt
     sind, wenn
     SETZEB(l,UEBRIG,*) angewendet wuerde ;
end else
(* mindestens eine Variable in V1 ist eindeutig be-
```

```
            stimmt, Fall "isoliert" :  *)
            begin
               sei l ein Literal, der eindeutig als wahr bestimmt
               ist;
               (* nicht-deterministische Wahl [wvar] *)
               waehle eine Variable wv in wvar[l´] und waehle
               ein Vorkommen vpwv von wv und ein Vorkommen
               vnwv von wv´ an den Punkten von WALD[l´] ;
               LABELSAMMELN(WALD[l´],vpwv,vnwv,w) ;
               AUFROLLEN ;
               if w#{} then
               begin
                  sei ll der Literal von w, der zuletzt in GE-
                  WAEHLT aufgenommen wurde ;
                  erweitere WALD[ll] durch eine Kante ka von ll
                  nach l´ ;
                  label[ka] := w -{ll} ;
               end else s := s + {l} ;
            end;
            SETZEB(l,UEBRIG,*);
            Il := Il + I[l] ;
            Vl := Vl - {Variable, die zu l gehoert} ;
         end; (* Ende der while-Schleife Vl *)
         VEREINFACHE(s) ;
      end;
      if UNERFUELLBAR then write(´ unerfuellbar ´)
      else write(´ erfuellbar ´);
end.
```

Beispiel VG:

Eingabe-KNF :

```
1.  a b c
2.        d e f
3.              g h i
4.                    j k l
5.  a´    d´
6.  a´          g´
7.  a´                j´
8.        d´    g´
9.        d´          j´
10.             g´    j´
11.   b´    e´
12.   b´          h´
13.   b´                k´
14.         e´    h´
15.         e´          k´
16.               h´    k´
17.     c´    f´
18.     c´          i´
19.     c´                l´
20.         f´    i´
21.         f´          l´
22.               i´    l´
```

1. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt der Fall "waehle" ein. Annahme : a wird gewaehlt. Bei der Ausfuehrung von SETZEB(a,UEBRIG,*) am Schluss der while-Schleife V1 wird der Baum WALD[a] aufgebaut.

Wir stellen einen Baum folgendermassen dar : Eine Kante wird durch eine Folge von Sternen repraesentiert. Wenn eine Kante eine nicht leere Labelmenge hat, unterbrechen wir in der Mitte die Folge von Sternen und schreiben dort die Labelmenge hin. Die Kanten sind von links nach rechts oder von oben nach unten gerichtet. WALD[a] hat folgende Gestalt :

```
a **** d´
**
* *
*  *
*   *
j´    g´
```

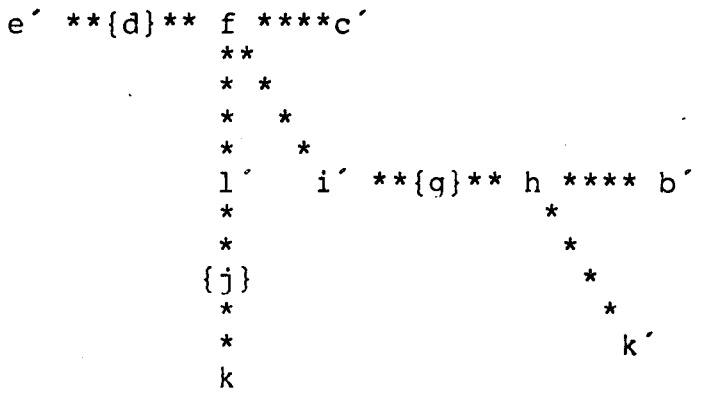
D.h. I[a] = {d´,g´,j´}. Die resultierende KNF UEBRIG sieht folgendermassen aus:
(Markierte Literale sind durch * ersetzt. Die Literale von gestrichenen Klauseln sind durch - ersetzt.)

```
-   - - -
2.          * e f
3.                * h i
4.                      * k l
-   *       -
-   *               -
-   *                       -
-           -       -
-           -               -
-                   -       -
11.    b´      e´
12.    b´              h´
13.    b´                      k´
14.            e´      h´
15.            e´              k´
16.                    h´      k´
17.       c´      f´
18.       c´              i´
19.       c´                      l´
20.             f´      i´
21.             f´              l´
22.                     i´      l´
```

In UEBRIG sind nicht alle Klauseln gestrichen, deshalb wird die for-Schleife EVAL nochmals ausgefuehrt. Jetzt tritt der Fall "unerfuellt" ein, und zwar sind die Variabeln e,f,h,i,k und l lernend. WALD[e] sieht z.B. folgendermassen aus :

```
        e**** b´
        **
        * *
        *  *
        *   *
        k´   h´ **{g}** i **** c´
        *               **
        *               * *
       {j}              *  *
        *               *   *
        *               l´   f´
        l
```

D.h. nach der Ausfuehrung von SETZEB(e,UEBRIG],WID]) in der for-Schleife EVAL ist wvar[e] = 1 und WID] ist wahr. WALD[e´] sieht z.B. folgendermassen aus :

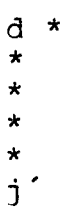


D.h. nach der Ausfuehrung von SETZEB(e´,UEBRIG2,WID2) ist wvar[e´] = k und WID2 ist wahr. Weil sowohl WID1 und WID2 wahr sind, ist die Variable e lernend.

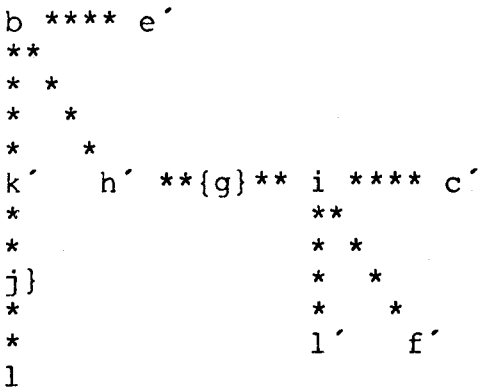
Der Aufruf der Prozedur LABELSAMMELN(e,l,l´,w1) liefert die Menge w1 = {g,j}. Der Aufruf von LABELSAMMELN(e´,k,k´,w2) liefert die Menge w2 = {d,g,j}. Deshalb ist w = w1 + w2 = {d,g,j}. Der Aufruf von AUFROLLEN liefert die Menge w = {a´}. Also ist a´ die (einzige) Klausel, welche in res[e] aufgenommen wird. Bei der Ausfuehrung der Anweisungen, die zum Fall "unerfuellt" gehoeren, wird die Klausel {a´} in sres aufgenommen. Wir nehmen an, dass diese Klausel bei der nicht-deterministischen Wahl [sres] gewaehlt wird. Also wird die Eingabe-KNF s um die Klausel {a´} erweitert. Nun wird die while-Schleife V1 verlassen, weil UEBRIG nach der Ausfuehrung von SETZEB(l,UEBRIG,*) keine Klausel mehr enthaelt. Nach der Ausfuehrung von VEREINFACHE(s) sieht s folgendermassen aus :

```
1.    b c
2.        d e f
3.              g h i
4.                    j k l
8.        d´    g´
9.        d´          j´
10.             g´    j´
11.   b´    e´
12.   b´          h´
13.   b´                k´
14.         e´    h´
15.         e´          k´
16.               h´    k´
17.     c´    f´
18.     c´          i´
19.     c´                l´
20.           f´    i´
21.           f´          l´
22.                 i´    l´
```

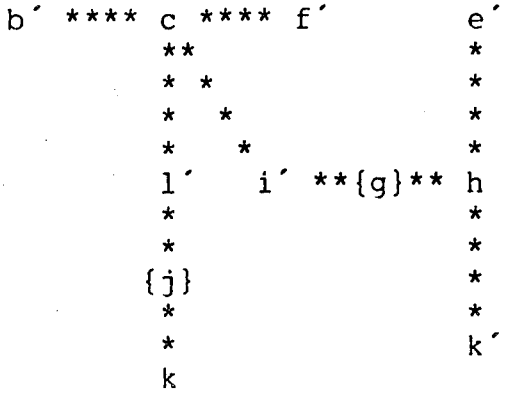
2. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt der Fall "waehle" ein. Annahme : d wird gewaehlt. Bei der Ausfuehrung von SETZEB(d,UEBRIG,*) wird WALD[d] folgendermassen aufgebaut :



In der resultierenden KNF UEBRIG sind nicht alle Klauseln gestrichen. Deshalb wird die for-Schleife EVAL nochmals ausgefuehrt. Jetzt tritt der Fall "unerfuellt" ein, und zwar sind die Variabeln b,c,h,i,k und l lernend. WALD[b] sieht z.B. folgendermassen aus :



Wald[b´] hat z.B. folgende Form :



Der Aufruf der Prozedur LABELSAMMELN(b,l,l´,wl) liefert die Menge wl = {g,j}. Der Aufruf von LABELSAMMELN(b´,k,k´,w2) liefert w2 = wl. Also ist w = {g,j}. Der Aufruf von AUFROLLEN liefert w = {d´}. Also ist {d´} eine Superresolvente. Nach der Ausfuehrung von VEREINFACHE sieht s folgendermassen aus :

```
1.     b c
2.           e f
3.                g h i
4.                      j k l
10.               g´    j´
11.    b´     e´
12.    b´           h´
13.    b´                 k´
14.          e´    h´
15.          e´          k´
16.                h´    k´
17.      c´    f´
18.      c´          i´
19.      c´                l´
20.           f´    i´
21.           f´          l´
22.                 i´    l´
```

3. Schritt :
Nachdem die for-Schleife EVAL ausgefuehrt ist, tritt bereits der Fall "unerfuellt" ein. Weil die Menge GEWAEHLT leer ist, wird die leere Superresolvente gebildet.

Der Superresolutionsbeweis wurde also nach drei missglueckten Modellkonstruktionen gefunden. Er hat folgende Gestalt:
(In Klammer steht eine lernende Variable.)

```
1.
2.

 .
 .     Eingabe-KNF
 .

21.
22.
23. a´ (e)
24. d´ (b)
25.    (b)
```

Kommentar zum Aufwand der Beweisueberpruefung:
Jeder Beweisschritt (Erzeugung einer Superresolvente) ist auf einer DZM (Direktzugriffmaschine) linear in der Laenge der momentanen KNF s ueberpruefbar. Man verwendet dazu die Prozedur SETZE. Bei der wiederholten Ausfuehrung dieser Prozedur wird jeder Literal von s hoechstens zweimal bearbeitet, d.h. aus seiner Klausel entfernt. Um die Ueberlegenheit von Superresolution gegenueber Resolution am Beispiel VG zeigen zu koennen, geben wir einen Resolutionsbeweis fuer die Klausel {a´} an, der nicht stark verkuerzt werden kann. Der ganze Resolutionsbeweis waere zu lang. Hinter jeder Resolvente stehen in Klammern die Nummern der Elternklauseln.

```
1.  a b c
2.        d e f
3.              g h i
4.                    j k l
5.  a´    d´
6.  a´          g´
7.  a´                j´
8.        d´    g´
9.        d´          j´
10.             g´    j´
11.   b´   e´
12.   b´         h´
13.   b´               k´
14.       e´    h´
15.       e´          k´
16.             h´    k´
17.     c´   f´
18.     c´         i´
19.     c´               l´
20.       f´    i´
21.       f´          l´
22.             i´    l´
23.           g h         l´   (3,22)
24.       e´  g           l´   (14,23)
25.       e´        j     l    (4,15)
26.       e´  g     j          (24,25)
27. a´    e´        j          (6,26)
28. a´    e´                   (7,27)
29.           g   i   k´       (3,16)
30.         f´g       k´       (20,29)
31.         f´      j k        (4,21)
32.         f´g     j          (30,31)
33.       d e   g     j        (2,32)
34. a´      e   g     j        (5,33)
35. a´      e         j        (6,34)
36. a´      e                  (7,35)
37. a´                         (28,36)
```

1.3 Der Zusammenhang zwischen SR und Superresolution

Ich zeige in diesem Abschnitt, dass jede von SR erzeugte Klausel im Fall einer missglueckten Modellkonstruktion eine Superresolvente ist. SR kann aber im allgemeinen fuer eine KNF s nicht alle moeglichen Superresolventen von s erzeugen. Das ist kein Nachteil von SR, sondern ein wesentlicher Vorteil, denn SR erzeugt nur gewisse "kurze" Superresolventen. Eine grosse Menge von Ballast-Superresolventen kann nicht erzeugt werden. Trotzdem ist SR, als Beweissystem aufgefasst, vollstaendig. Deshalb garantiert die Anwendung von SR auf eine unerfuellbare KNF, dass ein relativ kurzer, ballastfreier Superresolutionsbeweis entsteht.

Lemma 1.3.1 : Jede von Algorithmus SR in die Menge sres eingebrachte Klausel ist eine Superresolvente von s, d.h. jede von SR bei einer missglueckten Modellkonstruktion erzeugte Klausel ist eine Superresolvente.

Beweis:
Ich beziehe mich auf Algorithmus SR, Fall "unerfuellt". Sei sr eine Klausel in sres. Wenn genau die Literale in sr mit SETZE so gesetzt sind, dass sr unerfuellt ist, dann ist v lernend. Seien alle Literale in sr mit Hilfe von SETZE so gesetzt, dass sr unerfuellt ist. Es gebe eine unerfuellte Klausel in s. Dann kann sr nicht in sres sein, weil sr einen Literal enthaelt, der nicht in GEWAEHLT ist. Widerspruch. #

Das folgende Lemma zeigt, dass SR gewisse uninteressante Superresolventen nicht erzeugen kann.

Lemma 1.3.2 : Zu einer gegebenen KNF s koennen die Superresolventen in der Ausnahmemenge A, die im folgenden definiert wird, nicht von SR erzeugt werden.

Eine Superresolvente sr ist in A, gdw. es im Verlauf der Konstruktion von sr einen noch nicht gesetzten Literal l in der Literalmenge von s - sr gibt, so dass folgendes gilt :
1. Wenn l erfuellt wird, entsteht eine unerfuellte Klausel beim Anwenden von SETZE, d.h. l´ ist isoliert.
2. Sei I[l´] die Menge der durch l´ eindeutig bestimmten Literale. Reduziere s, indem nacheinander fuer alle Literale ll in I[l´] die Prozedur REDUZIERE(s,ll) ausgefuehrt wird. Dann darf die reduzierte Klausel sr im reduzierten s keine Superresolvente der reduzierten KNF mehr sein. #

Beispiel einer Superresolventen
in der Ausnahmemenge A :

1. a b c
2. a´ d e
3. a´ d e´
4. a b´ c
5. a f
6. a f´

7. c d

{c,d} ist eine Superresolvente der KNF s, bestehend aus den Klauseln 1 bis 6. {c,d} liegt aus folgendem Grunde in A : Wenn l = a erfuellt wird, entsteht beim Anwenden von SETZE eine unerfuellte Klausel. Also muss l´ = a´ erfuellt werden. {c,d} ist aber keine Superresolvente mehr von s, nachdem REDUZIERE(s,a´) auf s angewendet wurde.

Beweis von Lemma 1.3.2 :
Sei sr eine Superresolvente von s, die in A ist. Dann kann sr nicht von SR erzeugt werden, weil SR jede isolierte Variable feststellt.

Es gibt noch weitere Superresolventen, die von SR nicht erzeugt werden koennen. Eine Superresolvente sr heisst verfremdet, gdw. es eine Teilmenge t von sr gibt, so dass gilt:
1. t ist eine Superresolvente von s.
2. Wenn t mit SETZE unerfuellt gesetzt wird, ist mindestens ein Literal l in sr nicht eindeutig bestimmt (falls sr nicht in s vorhanden ist). Wir nennen l einen fremden Literal der Superresolventen sr.

Beispiel einer verfremdeten Superresolventen:
Wir verwenden die KNF s des Beispiels VG vom letzten Abschnitt. {a´,b} ist eine verfremdete Superresolvente von s, denn {a´} ist auch eine Superresolvente von s. Auch wenn a wahr gesetzt wird, ist die Variable b nicht eindeutig bestimmt. Deshalb ist b ein fremder Literal der Superresolventen {a´,b}. Man beachte, dass SR die Superresolvente {a´,b} nicht erzeugen kann.

Es bleibt spaeteren Untersuchungen vorbehalten, zu ueberpruefen, unter welchen Umstaenden SR nur nicht verfremdete Superresolventen erzeugt, und SR allenfalls so abzuaendern, dass er nur nicht verfremdete Superresolutionsbeweise erzeugt. Diese Untersuchungen sind deshalb interessant, weil das Beweissystem "nicht verfremdete Superresolution" (keine Superresolvente darf verfremdet sein) vollstaendig ist, die Beweise jedoch kuerzer sind.

In den Abschnitten 1.10 und 1.11 werden unabhaengige und maximal gekuerzte Superresolventen untersucht. Beide Begriffe be-

zeichnen, wie der Begriff "nicht verfremdet", eine Einschraenkung auf kuerzere Superresolventen. Wir nennen eine Superresolvente technisch minimal, wenn sie die folgenden drei Eigenschaften besitzt: nicht verfremdet, unabhaengig und maximal gekuerzt. Beim Beweissystem "technisch minimale Superresolution" TMSR muss jede Superresolvente technisch minimal sein. TMSR ist vollstaendig, obwohl es eine starke Einschraenkung von Superresolution ist. Zudem sind aber die Beweise in TMSR im allgemeinen kuerzer als im Beweissystem Superresolution, und technisch minimale Superresolventen koennen mit kleinem polynomialem Aufwand konstruiert werden. Deshalb ist TMSR ein sehr attraktives Beweissystem.

Bemerkung zum Aufwand der Konstruktion einer Superresolventen:

Ein Schritt von SR (Konstruktion einer Interpretation) liefert eine Superresolvente ausser, wenn in diesem Schritt ein Modell gefunden wird. Dieser Fall ist jedoch selten. Der Aufwand von SR auf einer DZM (Direktzugriffmaschine) fuer einen Schritt ist - grob abgeschaetzt - beschraenkt durch 1*1*1, wobei 1 die Literalanzahl der Eingabe-KNF ist. Diese obere Schranke ist unabhaengig von der Wahl der Varianten bei den nicht-deterministischen Anweisungen. (Es koennen auch Algorithmen fuer das Erzeugen von Superresolventen mit Aufwand O(1) angegeben werden, was vor allem fuer Anwendungen wichtig ist. Man beachte, dass SR mehr leistet, als was zur Erzeugung einer Superresolventen notwendig ist.) Also ist auf einer DZM ein Schritt von SR polynomial in der Laenge der behandelten KNF und somit ist auch das Erzeugen einer Superresolventen polynomial moeglich. Deshalb ist das Erzeugen eines Superresolutionsbeweises mit m Superresolventen und urspruenglichen Klauseln auf einer DZM polynomial in m.

1.4 Der Zusammenhang zwischen Resolution und Superresolution

In diesem Abschnitt zeige ich zuerst, dass jede von SR erzeugte Superresolvente eine Resolvente ist. Die Anzahl der Resolventenbildungen, die man benoetigt, um eine Superresolvente zu erhalten, wird nach oben abgeschaetzt. Daraus folgt, dass SR polynomial auf Resolution reduzierbar ist. Weil ein beliebiger Superresolutionsbeweis auf einen von SR erzeugbaren Superresolutionsbeweis verkuerzt werden kann, gilt: Superresolution ist polynomial reduzierbar auf Resolution.

Lemma 1.4.1 : Jede von SR erzeugte Superresolvente einer KNF s ist gleich dem Resultat einer Folge von Resolutionsoperationen.

Beweis:
Sei w die Menge von Literalen, die von LABELSAMMELN(l,vp,vn,w) berechnet wird. Dann ist die Menge w+{l} eine Resolvente von s, wobei bei jeder Resolventenbildung genau eine Elternklausel in s ist. Man bemerke, dass an jeder Kante ka eines Baumes in WALD im wesentlichen eine Klausel von s gespeichert ist. Sei h1 der Literal des Anfangspunktes, h= label[ka] und h2 der Literal des Endpunktes. Dann entspricht dieser Kante die Klausel c={h1´,h,h2}. Ebenso kann die Berechnung von AUFROLLEN mit Resolution simuliert werden. Auch hier ist bei jeder Resolventenbildung genau eine Elternklausel in s. #

Sei sr eine von SR erzeugte Superresolvente und v eine lernende Variable. Sei m die Anzahl Klauseln in s. Zur Bildung von sr + {v} werden hoechstens m Resolutionen benoetigt, denn jede Klausel in s kann hoechstens einmal als Elternklausel vorkommen. (Die Baeume in WALD mit Indizes in GEWAEHLT enthalten jede Klausel hoechstens einmal.)

Analoges gilt fuer sr + {v´} . Also sind zur Bildung von sr hoechstens 2*m + 1 Resolutionen noetig. Bei genau einer Anwendung von Resolution ist keine Elternklausel in s.

Lemma 1.4.2 : Sei s eine unerfuellbare KNF mit m1 Klauseln, und sei m die Anzahl Klauseln (inklusive die Klauseln in s) in einem von SR erzeugten Superresolutionsbeweis von s. Dann kann in polynomialer Zeit, abhaengig von der Laenge von s und des Superresolutionsbeweises, ein Resolutionsbeweis mit O(m*m) Resolutionen angegeben werden.

Beweis:
Die Anzahl Resolutionen ist beschraenkt durch

$$\sum_{k=0}^{m-m1-1} (2*(m1+k)+1).$$

#

Sei sr eine beliebige Superresolvente einer unerfuellbaren KNF s. Wir wenden SR auf s an und setzen dabei die Literale in sr unerfuellt. Sei sr1 die Superresolvente, die dabei von SR erzeugt werden muss. Es ist moeglich, dass sr1 eine Teilmenge von sr ist. Mit dieser Methode kann jeder Superresolutionsbeweis von hinten (beginnend mit der leeren Superresolventen) auf einen Beweis, der durch SR erzeugbar ist, verkuerzt werden. Deshalb haben wir bewiesen:

Satz 1.4.3 : Superresolution ist polynomial reduzierbar auf Resolution.

1.5 Die implizite Leistung von Superresolution

Obwohl ich spaeter beweisen werde, dass Resolution und Superresolution polynomial aequivalent sind, zeige ich in diesem Abschnitt, dass von einem anderen Gesichtspunkt her Superresolution exponentiell besser ist als Resolution.

Sei s eine erfuellbare KNF und R(s) die Menge aller Resolventen von s. Auf den "meisten" Formelmengen ist die Anzahl der Klauseln in R(s) eine exponentielle Funktion bezueglich der Klauselanzahl von s. Es wird sich zeigen, dass die Menge EE(s) der Klauseln in R(s), die keine Superresolventen von s sein koennen, gross ist. Jede KNF s kann polynomial so transformiert werden, dass EE(s) exponentiell in der Laenge von s waechst. Also kann man sagen, dass Superresolution exponentiell weniger Wahlen als Resolution fuer die Fortsetzung eines Beweises offen laesst. Deshalb beruecksichtigt SR, wenn ein Literal l gewaehlt wird, gleichzeitig exponentiell viele Resolventen im folgenden Sinn: Keine dieser Resolventen kann unerfuellt werden, wenn l und die durch l eindeutig bestimmten Literale wahr gesetzt werden.

Definition 1.5.1 : Sei s eine KNF. Dann heisst eine Klausel c durch einseitige Resolution erzeugt, wenn c in der folgenden Menge E(s) liegt:

1. Alle Klauseln in s sind in E(s).

2. Alle Resolventen von zwei Klauseln c1,c2 sind in E(s), falls c1 in E(s) und c2 in s ist.

3. Es gibt keine andern Klauseln in E(s).

Eine Klausel c heisst durch einseitige Resolution mit Einerklauselelimination erzeugt, wenn c in folgender Menge EE(s) liegt:

1. Alle Klauseln in E(s) sind in EE(s).

2. Sei l eine Klausel der Laenge 1, welche in EE(s) ist. Sei neg(l) die Menge der Klauseln in EE(s), welche den Literal l´ enthalten. Sei red die Menge von Klauseln, die erhalten wird, wenn bei jeder Klausel in neg(l) der Literal l´ weggelassen wird. Dann ist red auch in EE(s).

3. Es gibt keine andern Klauseln in EE(s).

Bemerkung :
E(s) ist die Menge aller durch Input-Resolution [CH73] erzeug-

baren Klauseln. Man kann zeigen: E(s) enthaelt die leere Klausel genau dann, wenn ein Resolutionsbeweis fuer s existiert, in dem bei jedem Schritt mindestens eine Elternklausel die Laenge 1 hat.

Lemma 1.5.1 : Sei s eine KNF und EE(s) die Menge der durch einseitige Resolution mit Einerklauselelimination erhaltenen Resolventen. Wenn EE(s) die leere Klausel nicht enthaelt, kann keine Klausel in EE(s) durch SR erzeugt werden. Wenn EE(s) die leere Klausel enthaelt, so erzeugt SR die leere Klausel im ersten Schritt.

Beweis:
1. Die leere Klausel sei nicht in E(s). Wir zeigen zuerst, dass keine Superresolvente in E(s) sein kann. Wir beweisen folgendes mit Induktion ueber die Struktur von E(s): Wenn eine Klausel in E(s) unerfuellt ist, so ist nach der Anwendung von SETZE eine Klausel in s unerfuellt. Die Aussage ist richtig fuer Klauseln in s. Sei c1 eine Klausel in E(s) und c2 eine Klausel in s. Sei r die Resolvente von c1 und c2.
a) r ist leer. Dies ist nicht moeglich, denn sonst waere die leere Klausel in E(s).
b) r ist nicht leer. Wenn r unerfuellt ist, so ist entweder c1 oder c2 als unerfuellt bestimmt. Wenn c2 unerfuellt ist, so ist eine Klausel in s unerfuellt. Wenn c1 unerfuellt ist, so wenden wir die Induktionsvoraussetzung an.

Also kann keine Klausel in E(s) eine Superresolvente sein. Wegen Lemma 1.3.1 kann somit keine Klausel in E(s) von SR erzeugt werden.

2. Die leere Klausel sei nicht in EE(s). Wir zeigen, dass SR keine Superresolvente in EE(s) erzeugt unter der bereits bewiesenen Annahme, dass keine Superresolvente in E(s) ist. Wir verwenden Induktion ueber die Struktur von EE(s). Fuer E(s) ist die Behauptung bewiesen. Sei l eine Klausel der Laenge 1 in EE(s). Dieser eindeutig bestimmte Literal wird von SR erkannt und stets erfuellt, falls das Komplement nicht auch eindeutig bestimmt ist. Sei c2 eine Klausel in EE(s), welche den Literal l' enthaelt. Sei r die Resolvente von l und c2.
a) r ist leer. Dies ist nicht moeglich, denn sonst waere die leere Klausel in EE(s).
b) r ist nicht leer. Wenn r unerfuellt ist, so ist entweder l oder c2 unerfuellt. Da l nicht unerfuellt sein kann, ist stets c2 unerfuellt. Also kann r nicht von SR erzeugt werden, da sonst die Induktionsannahme verletzt waere.

Wenn E(s) die leere Klausel enthaelt, gilt folgendes: Sei v die Variable, die aufgeloest wurde, als die leere Klausel entstand. Wenn v von SR gesetzt wird, entsteht bei beiden Interpretationen eine unerfuellte Klausel. GEWAEHLT ist aber zu Beginn von SR leer, deshalb erzeugt SR die leere Klausel im ersten Schritt.

Wenn EE(s) die leere Klausel enthaelt, werden die eindeutig bestimmten Literale im ersten Schritt von SR wahr gesetzt. Fuer die resultierende reduzierte KNF s´ gilt E(s´)=EE(s´). Deshalb erzeugt SR die leere Klausel im ersten Schritt. #

Korollar 1.5.2 : Man braucht mindestens 3 Anwendungen von Resolution, um irgendeine Klausel (ausser die leere) zu erhalten, die mittels Superresolution mit einer einzigen Anwendung der Schlussregel erzeugbar ist.

Beweis:
Verwende die Tatsache, dass E(s) keine Superresolventen (ausser allenfalls die leere) enthaelt.
#

Korollar 1.5.3 : Sei s eine erfuellbare KNF. Dann erzeugt SR die Menge EE(s) in folgendem Sinn implizit: SR waelt die Interpretation l der ersten zu setzenden Variabeln so, dass keine Klausel in EE(s) unerfuellt wird, wenn l und die durch l eindeutig bestimmten Literale gesetzt werden.

Beweis:
Annahme: Sei c eine Klausel in EE(s), die bei der ersten Anwendung von SETZE unerfuellt wird. Nach Beweis von Lemma 1.5.1 ist dann eine Klausel in s unerfuellt. Dies ist bei Algorithmus SR nur moeglich, wenn eine lernende Variable existiert. Wir erhalten einen Widerspruch zur Voraussetzung, dass s erfuellbar ist. #

Das Folgende soll darauf hinweisen, dass EE(s) normalerweise eine sehr grosse Menge ist.

Sei s eine erfuellbare KNF mit m verschiedenen Klauseln c(1),c(2), ... ,c(m). Wir betrachten folgenden Algorithmus, der eine Teilmenge En(s) von E(s) konstruiert :

```
for k:=1 to m do
begin
   bilde die Menge R der Resolventen zwischen c(k) und allen
   andern Klauseln in s ;
   erweitere s um R;
   eliminiere Klauseln in s, die mehr als einmal vorkommen,
   bis die Klauseln in s nur einmal auftreten;
end;
```

Lemma 1.5.4 :
1. Wenn R fuer ein k leer ist, so ist c(k) ohne Einfluss auf die Modelle von s.
2. En(s) - s enthaelt alle wichtige Information ueber s,

denn es gilt : En(s) - s ist erfuellbar, gdw. s erfuellbar ist. Es kann jedoch Modelle von En(s) - s geben, die keine Modelle von s sind. Jedes derartige Modell laesst sich jedoch leicht in ein Modell von s abaendern.

Beweis :
Siehe [ME68].

Obiger Algorithmus simuliert vollstaendig die Elimination einer beliebigen Variabeln in der Davis-Putnam-Prozedur [DA60]. Die weiteren Variabelneliminationen werden nur fragmentarisch simuliert. Fuer die Davis-Putnam-Prozedur ist bewiesen, dass sie schon auf einfachen Formeln exponentiell ist [TS68,KIR74]. Das ist ein Hinweis, dass En(s) und somit auch EE(s) normalerweise gross sind.

Definition 1.5.2 : Sei s eine erfuellbare KNF, auf die SR angewendet wird. Sei l der erste Literal, der von SR im ersten Schritt gewaehlt wird. Wir definieren VS(s) als die Anzahl der Resolventen von s, die nicht unerfuellt sind, bevor der naechste Literal gewaehlt wird.

Definition 1.5.3 : Sei s eine erfuellbare KNF mit m verschiedenen Klauseln und n Variabeln. s heisst normal, wenn card(EE(s)) $\geq$ 2**(n*1/3).

Wenn eine KNF s mit m Klauseln und n Variabeln nicht normal ist, kann sie auf natuerliche Art und mit polynomialem Aufwand folgendermassen in eine normale KNF verwandelt werden: Suche eine Interpretation I von s, die kein Modell ist. Erweitere s um eine Klausel c, in der jeder Literal durch I unerfuellt ist. Teile die Literale in c in n/2 disjunkte Literalpaare ein und fuehre fuer jedes dieser Paare eine Hilfsvariable ein. Die so erweiterte KNF s enthaelt nun n1 = n + n/2 Variabeln. E(s) enthaelt mindestens 2**(n1*1/3) = 2**(n/2) verschiedene Klauseln, denn die Klausel c kann mit einseitiger Resolution auf 2**(n/2) verschiedene Arten gekuerzt werden. Somit ist bewiesen:

Lemma 1.5.5 : Fuer alle erfuellbaren KNF s, die normal sind, ist VS(s) eine exponentielle Funktion in der Klauselanzahl von s.

SR hat fuer erfuellbare KNF die Eigenschaft, im Normalfall mit polynomialem Aufwand eine exponentielle Anzahl von Resolventen gleichzeitig zu beruecksichtigen, d.h. keine dieser Resolventen wird unerfuellt, wenn der naechste Literal und die durch

ihn eindeutig bestimmten Literale gesetzt werden. Von diesem Gesichtspunkt her kann man sagen, dass Superresolution exponentiell besser ist als Resolution.

Korollar 1.5.6 : Fuer die unerfuellbaren KNF s, fuer welche die leere Klausel in EE(s) liegt, kann effektiv ein polynomialer Resolutionsbeweis angegeben werden.

Beweis:
Verwende Saetze 1.4.2 und 1.5.1. #

Wir betrachten im folgenden Resolution und Superresolution als Schlussregelsysteme fuer UNERF. Sei UNERFNT die Menge der unerfuellbaren KNF s, bei denen fuer keine Variable v die Klausel v und die Klausel v´ in s vorkommt. Sei T ein Schlussregelsystem fuer UNERF, und sei s eine KNF. Dann ist T*(s) die Menge aller Klauseln, welche mit T aus s herleitbar sind.

Ein Schlussregelsystem T1 heisst eine Verbesserung des Schlussregelsystems T2, wenn fuer alle s in UNERFNT die Menge T1*(s) eine echte Teilmenge von T2*(s) ist [ME71]. Wir nennen einen Superresolutionsbeweis normal, wenn er von SR erzeugt wurde.

Lemma 1.5.7 : Normale Superresolution ist eine Verbesserung von Resolution.

Beweis :
Nach dem Beweis des Lemmas 1.5.1 gibt es fuer KNF in UNERFNT stets Resolventen, die keine Superresolventen sein koennen.

1.6 Resolution ist polynomial verkuerzbar auf Superresolution

In diesem Abschnitt zeige ich, dass Superresolution, ausser in trivialen Faellen, stets kuerzere Beweise erlaubt als Resolution, d.h. Beweise, die weniger Klauseln enthalten. Ein Beweissystem BW1 heisst polynomial verkuerzbar auf ein Beweissystem BW2, wenn fuer alle KNF s in UNERFNT gilt :

1. Zu jedem Beweis w1 von s in BW1 kann in polynomialer Zeit abhaengig von der Laenge von w1, ein Beweis w2 in BW2 fuer s angegeben werden.
2. Die Laenge von w2 (im Komplexitaetsmass des Beweissystems BW2) ist kuerzer als die Laenge von w1 (im Komplexitaetsmass des Beweissystems BW1).

Lemma 1.6.1 : Sei s eine KNF und E(s) die Menge der durch einseitige Resolution erhaltenen Resolventen. Wenn E(s) die leere Klausel nicht enthaelt, dann kann keine Klausel in E(s) eine Superresolvente sein.

Beweis:
Analog zum Beweis von Lemma 1.5.1 (1.Teil).

Lemma 1.6.2 : Sei s eine KNF und r(1), r(2), ... r(k)={} ein Resolutionsbeweis fuer s, der nicht trivial ist. Dann kann in polynomialer Zeit, abhaengig von der Laenge von s und des Beweises, ein kuerzerer Superresolutionsbeweis fuer s angegeben werden. d.h. Resolution ist polynomial verkuerzbar auf Superresolution (Komplexitaetsmass: Anzahl Resolventen, resp. Superresolventen).

Beweis:
Wir betrachten einen Abkuerzungsalgorithmus fuer r(1), r(2), ... r(k)={}. Sei r(j) (j zwischen 1 und k) die erste Klausel mit folgender Eigenschaft W: Wenn sie mit SETZE unerfuellt gesetzt wird, ist keine Klausel in s unerfuellt. r(j) muss aus folgenden Gruenden eine Superresolvente von s sein:

a) Wenn mit SETZE r(j) unerfuellt gesetzt wird, ist keine Klausel in s unerfuellt.
b) Wenn r(j) unerfuellt gesetzt ist, muss es eine lernende Variable v geben, naemlich die Variable, die aufgeloest wurde, als durch Resolution r(j) entstand. Denn wenn diese Variable gesetzt wird, muss bei beiden Interpretationen eine unerfuellte Klausel entstehen. Man beachte, dass r(j) die erste Klausel im Beweis ist, fuer die W gilt.

Falls r(j)#{}, so erweitere s um r(j) und wiederhole den Algorithmus so lange, bis j=k. Die ausgewaehlten Resolventen sind Superresolventen. Somit haben wir einen Superresolutionsbeweis

konstruiert. Weil k>1 und wegen Lemma 1.6.1, ist der Superresolutionsbeweis kuerzer als der gegebene Resolutionsbeweis. #

Betrachten wir folgenden Resolutionsbeweis :

1.	a	b	c				
2.	a	b	c´				
3.	a	b´		d			
4.	a	b´		d´			
5.	a´	b			e		
6.	a´	b			e´		
7.	a´	b´				f	
8.	a´	b´				f´	
9.	a	b					(1,2)
10.	a	b´					(3,4)
11.	a						(9,10)
12.	a´	b					(5,6)
13.	a´	b´					(7,8)
14.	a´						(12,13)
15.							(11,14)

Die KNF, deren Unerfuellbarkeit bewiesen wird, besteht aus den Klauseln 1 bis 8. Der Resolutionsbeweis besteht aus den Klauseln 9 bis 15. Die erste Superresolvente in diesem Beweis ist die Klausel 11 und die zweite die Klausel 15.

Lemma 1.6.4 : Resolution und Superresolution sind polynomial aequivalente Beweissysteme fuer UNERF.

Beweis:
Resolution < Superresolution (Lemma 1.6.2)
Superresolution < Resolution : verwende Lemma 1.4.3.
#

Satz 1.6.5 : Auch die folgenden Beweissysteme, die Abarten von Resolution sind, sind polynomial verkuerzbar auf Superresolution (Komplexitaetsmass: Anzahl Klauseln, d.h. Anzahl semantischer Resolventen, Anzahl Hyperresolventen etc.): semantische Resolution, Hyperresolution, set-of-support Resolution, Lockresolution, lineare Resolution, Resolution mit Mischen etc. (Fuer die Definitionen siehe [CH73].)

Beweis:
Verwende obigen Abkuerzungsalgorithmus.
#

ZITIERTE ARBEITEN

****

DIE VERWENDETEN ABKUERZUNGEN VON ZEITSCHRIFTENNAMEN SIND AUF DER ZWEITEN SEITE DER ANSCHLIESSEND FOLGENDEN BIBLIOGRAPHIE ERKLAERT.

AA76. AANDERAA S., HORN FORMULAS AND THE P=NP PROBLEM, LOGIC COLLOQUIUM 76, OXFORD.

AHO75. AHO A.V., HOPCROFT J.E., ULLMAN J.D., THE DESIGN AND ANALYSIS OF COMPUTER ALGORITHMS, ADDISON-WESLEY.

CH73. CHANG C.L., LEE R.C., SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING, ACADEMIC PRESS, NEW YORK AND LONDON, 1973.

CO71. COOK S.A., THE COMPLEXITY OF THEOREM-PROVING PROCEDURES, 3.STOC (1971), 151-158.

CO74. COOK S., RECKHOW R., ON THE LENGTH OF PROOFS IN THE PROPOSITIONAL CALCULUS, 6.STOC (1974), 135-148.

CO75. COOK S.A., FEASIBLY CONSTRUCTIVE PROOFS AND THE PROPOSITIONAL CALCULUS, 7.STOC (1975).

COO76. COOK S.A., A SHORT PROOF OF THE PIGEON HOLE PRINCIPLE USING EXTENDED RESOLUTION, SIGACT NEWS, VOL 8, NO 4.

DA60. DAVIS M., PUTNAM H., A COMPUTING PROCEDURE FOR QUANTIFICATION THEORY, J.ACM, VOL 7, 201-215.

GA74. GALIL Z., ON THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY, 1974.

GA75. GALIL Z., ON THE VALIDITY AND COMPLEXITY OF BOUNDED RESOLUTION PROCEDURES, 7.STOC (1975).

GAL76. GALIL Z., ON ENUMERATION PROCEDURES FOR THEOREM PROVING AND FOR INTEGER PROGRAMMING, IN AUTOMATA, LANGUAGES AND PROGRAMMING, THIRD INTERNATIONAL COLLOQUIUM, EDINBURGH, 1976.

HART76. HARTMANIS J., HOPCROFT J., INDEPENDENCE RESULTS IN COMPUTER SCIENCE, SIGACT NEWS, VOL 8, NO 4.

HU68. HU S., MATHEMATICAL THEORY OF SWITCHING CIRCUITS AND AUTOMATA, UNIVERSITY OF CALIFORNIA PRESS, BERKELEY AND LOS ANGELES.

MIL75. MILLER G.L., RIEMANN'S HYPOTHESIS AND TESTS FOR PRIMALITY, 7.STOC.

POS76. POSA L., HAMILTON CIRCUITS IN RANDOM GRAHS, DISCRETE MATHEMATICS 1976.

PR74. PRATT V.R., RABIN M., STOCKMEYER L.J., A CHARACTERIZATION OF THE POWER OF VECTOR MACHINES, 6.STOC (1974), 122-134.

REU76. REUSCH B., ON THE GENERATION OF PRIME IMPLICANTS, TECHNICAL REPORT, DEPARTMENT OF COMPUTER SCIENCE, CORNELL UNIVERSITY.

RAB76. RABIN M.O., PROBABILISTIC ALGORITHMS, RC 6164, IBM T.J.WATSON RESEARCH CENTER AND IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS.

RO65. ROBINSON J.A., A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM, VOL 12, NO 1, 23-41.

RO68. ROBINSON J.A., THE GENERALIZED RESOLUTION PRINCIPLE, MI 3,77-93.

SC76. SCHNORR C.P., OPTIMAL ALGORITHMS FOR SELF-REDUCIBLE PROBLEMS, THIRD INTERNAT. COLL. ON AUTOMATA, LANG.AND PROG., EDINBURGH.

SHO76. SHOSTAK R.E., ON THE COMPLEXITY OF RESOLUTION, IN ALGORITHMS AND COMPLEXITY, ED. J.F. TRAUB, ACADEMIC PRESS, P.493.

SOL76. SOLOWAY R., STRASSEN V., A FAST MONTE-CARLO TEST FOR PRIMALITY, SUBMITTED FOR PUBLICATION.

SP75. SPECKER E., STRASSEN V., KOMPLEXITAET VON ENTSCHEIDUNGSPROBLEMEN, LNCS 43.

TAR76. TARJAN R.E., TROJANOWSKI A.E., FINDING A MAXIMUM INDEPENDENT SET, STAN-CS-76-550, STANFORD UNIVERSITY.

TS68. TSEITIN G.S., ON THE COMPLEXITY OF DERIVATIONS IN THE PROPOSITIONAL CALCULUS, STRUCTURES IN CONSTRUCTIVE MATHEMATICAL LOGIC, PART II, A.O. SILENKO(ED), 1968, 115-125.

WI76. WIETLISBACH M., EINE UNTERE SCHRANKE FUER GENTZEN OHNE SCHNITT, PRIVATE MITTEILUNG.